Feasibility Study of a GNSS Tracking Application on Android

Tsailing Wong

University of Tampere
School of Information Sciences
Software Development
M.Sc. thesis
Supervisor: Paula Syrjärinne
September 2015

University of Tampere
School of Information Sciences
Software Development
Tsailing Wong: Feasibility Study of a GNSS Tracking Application on Android
M.Sc. thesis, 60 pages, 30 appendix and index pages
September 2015

Abstract

There is a need for better and more accessible tools in the field of location-based services. To address this problem, Here.com proposed a mobile application that allows the tracking of historic and future location of navigation satellites in orbit. The work for this thesis involves the feasibility analysis and implementation of a GNSS tracking application on Android mobile devices that displays historic satellite data in a time lapse slider interface. Considerations have to be made to accommodate the limitations of a mobile device, including limited storage capacity, the consequence of memory leaks, and download data costs. As with any scientific tool, algorithm accuracy and performance efficiency are of utmost importance. Therefore, the work for this thesis includes the study of algorithmic accuracy for GPS coordinate conversion and the analysis of system performance for the application on Android.

The result of this thesis is a solution which involves both a custom implementation and a combination of several open source code. The conclusion provides a few possible future researches to extend the results of this thesis, as well as possible future enhancements that can be made to improve the application.

Keywords: algorithm, Android, Cartesian and geodetic coordinates, GNSS, Java, mobile application, performance.

Acknowledgements

I would like to thank my supervisor Paula Syrjärinne for her help on the subject matter, as well as Dr Zhang Zheying for her guidance on writing this thesis. I would also like to thank Here.com for the provision of this project.

I would also like to thank my family for their support while I worked on my Master's degree, especially my aunts, Nancy Seow, Betty Ho, and Wendy Lee, for their continuously gracious hospitality and care throughout the years. And to my lovely sisters, Tsaiching and Tsaixing Wong, thank you guys for always being there for me and I love you both always.

In addition, I would like to thank Jennifer Chow and Brooke Treseder of Pentaho for their trust and confidence in my work, and putting up with my erratic schedule. Their continual patronage of my programming services has allowed me to enjoy the travel freedom that I have today.

Tampere, September 20th, 2015

Tsailing Wong

Contents

List of Abbreviations 1
List of Symbols 3
1. Introduction 4
1.1 Scope of the thesis 5
1.2 Thesis organization 6
1.3 Research Questions 6
1.4 Research Design and Methods 7
2 GNSS Concepts 9
2.1 Background 9
2.2 Navigation Satellite Systems 10
2.3 Coordinate Systems 10
2.4 Cartesian to Geodetic Coordinates Conversion Methods 13
2.5 Satellite Location Data from the International GPS Service (IGS) 17
3 Android Concepts 20
3.1 Android Platform Overview 20
3.2 Optimizing for Android 22
4 Requirements 26
4.1 Hardware Requirements 26
4.2 Basic Requirements for Mobile Applications 26
4.3 Functional Requirements 27
4.4 Non-Functional Requirements 29
5 Implementation 31
5.1 Tools and Technology 31
5.2 Software Architecture 31
5.3 User Interface 33
5.4 SQLite Database 37
5.5 Satellite Data Download from IGS 37
6 Evaluation 39
6.1 Benchmark Setup 39
6.2 Benchmark Tests 40
6.3 Benchmark Results and Analysis 42
6.4 Analysis of Implementation 55
6.5 Quality of Benchmark and Evaluation of Results 55
6.6 Failures and Limitations 56
7 Conclusions and Future Work 58
7.1 Answers to Research Questions 58
7.2 Conclusions 60
7.3 Future work 61

References 64
Appendix 1 – XYZ to LLH Coordinate Conversion Code 72
Appendix 2 – Open Sourced Code Modules 77
Appendix 3 – Google Maps API 78
Appendix 4 – Data Type Tests 81

List of Abbreviations

ADT	Android Development Tools
AOT	Ahead-Of-Time compilation for Android's ART runtime system
API	Application Programming Interface
ART	Android Runtime
CRS	Coordinate Reference System
ECEF	Earth-Cantered, Earth-Fixed
ESA	European Space Agency
EU	European Union
FAA	Federal Aviation Administration
FTP	File Transfer Protocol
GLONASS	Global Navigation Satellite System, Russia's version of GPS (Globalnaya Navigazionnaya Sputnikovaya Sistema)
GNSS	Global Navigation Satellite System
GPS	Global Positioning System (or the United States NAVSTAR GPS)
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
IGS	International GPS Service
JIT	Just-In-Time compilation for Android's Dalvik virtual machine
LLH	Latitude, longitude, and height (or altitude)
LZC	Lempel-Ziv-Thomas compression module
NAD-27	North American Datum 1927
NDK	Native Development Kit
NGA	National Geospatial-Intelligence Agency
OS	Operating System
PRN	Pseudo-Random Number
QZSS	Quasi-Zenith Satellite System
SBAS	Satellite-Based Augmentation System
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SNR	Signal-to-Noise Ratio
UI	User Interface
UTM	Universal Transverse Mercator
WAAS	Wide Area Augmentation System

WGS-84	World Geodetic System 1984
XML	Extensible Markup Language

List of Symbols

φ or ϕ	Geodetic latitude, positive north
λ	Geodetic longitude, positive east
h	Height above ellipsoid
a	Semi-major axis. This is the equatorial axis or radius of the ellipsoid. In the WGS-84 datum, this is represented as: $a = 6378137$ m
b	Semi-minor axis. This is the polar axis or radius of the ellipsoid. In the WGS-84 datum, this is represented as: $b = 6356752.31424518$ m
f	This is a ratio describing the amount of flattening on the ellipsoid. Flattening is represented by: $f = \frac{a-b}{a}$
e^2	First eccentricity of the ellipsoid. This is usually represented as e^2 in the following equation: $e^2 = \frac{a^2-b^2}{a^2}$
ε^2	Second eccentricity of the ellipsoid. This is usually represented as ε^2 in the following equation: $\varepsilon^2 = \frac{a^2-b^2}{b^2}$
p	Distance from minor axis, represented by the following equation: $p = \sqrt{x^2+y^2}$
N	Radius of curvature in the prime vertical. This is usually represented by the following equation: $N = \frac{a}{\sqrt{1-e^2\sin^2\varphi_1}}$
x, y, z	Cartesian coordinates in a geocentric coordinate reference system

1. Introduction

The United States NAVSTAR Global Positioning System (GPS) and the Russian Globalnaya Navigazionnaya Sputnikovaya Sistema (GLONASS) are global navigation satellite systems (GNSS) that provide location information on Earth. Each satellite continually transmits its satellite position and the time of transmission to ground GPS units [Astronautix.com, n.d.]. As the satellites orbit around Earth, a given position on Earth will observe a visibility of different satellites overhead. This affects the satellite data received by the GPS receivers on the ground [United States, 1996]. Therefore, mapping and location-based service companies such as Here.com have indicated a need for a variety of tools to monitor satellites. They need tools that can provide information on satellites, such as position coordinates and transmission health. It would be beneficial for the field if these tools were made more accessible, both by having the tools available on common devices and by having the source code of such tools openly available for customization. One such common device would be a mobile device. A mobile device is small and easy to carry and most people have at least one mobile device with them at all times. According to Mobile Marketing Statistics 2015, there are currently more than 1.5 billion mobile Internet users worldwide and roughly 80 per cent of Internet users own a smartphone [Smart Insights, 2015]. Therefore, it makes sense to have an open-sourced satellite tracking application built for mobile devices. This thesis describes the development process and solutions behind the GNSS Tracking Application for mobile devices. It also provides a study on the different algorithms used for GPS coordinate conversions, including a study on the accuracy and the computational costs of the methods.

The idea for this thesis started with Here.com, a Nokia subsidiary located in Tampere, Finland. The department which proposed the idea for the application is involved with creating radio maps. Tools that visualize past, present and future satellite positions would help them analyze past mappings and plan future mapping campaigns.

There are currently many real-time satellite-tracking applications on the mobile market. For example, the GNSS Radar application [iTunes, 2014] for iOS shows the current GNSS constellation on a sky plot. The P-Track Satellite Viewer [iTunes, 2015] for iOS displays past, present, and future locations of GPS, GLONASS, as well as some amateur satellites on a 3D interactive globe. These applications, however, are not open sourced. On the open source market, there is the YGPS Satellites application [Yunnanexplorer, n.d.] for Android, which shows the location and signal strength of the GPS satellites on a sky plot. There is also the GPSTest Android application [GitHub, 2015a] which displays basic information about the user's current GPS location, including the GPS signal strength, and the elevation and azimuth values of the satellites. It is useful for a user to see the GPS signal strength for their current location to gauge

the effectiveness of location-based applications on their device. Despite the myriad of GNSS-related applications on the mobile market, it is still missing an open-sourced application that displays a multi-day cache of historic satellite location data on a map.

This thesis makes a contribution to the field by providing an open-sourced GNSS Tracking application for the Android platform. The code is shared on GitHub at `https://github.com/UTA-Here/GNSSTAM` and available for download and customization. The GNSS Tracking application will not change the way people work, but will broaden the available options for those who wishes to have such an application for Android mobile devices. It provides a method for users to track the location of satellites over a period of several days via a time-lapse slider.

The challenges involved in dealing with satellite positioning data include the limited sources and unpredictable availability of historic satellite position data, discrepancies in calculations, as well as other factors that could contribute to data inaccuracies [United States, 1996]. In addition, the large amount of data involved in displaying a multi-day cache of satellite information is one that may render the application unusable, either due to performance issues or memory leaks. Considerations have to be made to accommodate the technical limitations of a mobile device, including limited storage capacity, consequence of memory leaks, and download data costs. All of these challenges will be addressed in research of this thesis.

It is prudent to keep in mind that device performance is only one of the many considerations of mobile application development. Other important requirements include the user experience and reliability of using the application, and the human resource needed to build and maintain the application [Wasserman, 2010]. All these will be important factors in determining if the implementation of such an application is feasible, and will help in deciding on the best solution for the GNSS Tracking application.

From here on, the abbreviation “app” will be used when referring to mobile applications.

1.1 Scope of the thesis

The work for this thesis includes a feasibility analysis on the implementation of the GNSS Tracking app for the Android mobile devices. In addition, the satellite location data retrieved from the third party source is of a different coordinate system than that which is required by the Google Maps API. Therefore, conversion needs to be performed on all the satellite coordinate data. As part of the research for this thesis, algorithmic accuracy and system performance analysis for GPS coordinate conversion

on mobile devices will be conducted. Performance and accuracy of four different algorithms for GPS coordinate conversion will be benchmarked and analysed.

1.2 Thesis organization

As the work for this thesis spans two traditional fields (GNSS and Android app development), there will be separate chapters dedicated to each of these areas. This is to acclimate readers who may not have any experience with some of the material needed to follow the thesis.

There are seven chapters in this thesis. Chapter 1 contains an introduction to the thesis, including the research questions, methods, and design. Chapter 2 opens the discussion with some background information on the GNSS concepts used in this thesis with focus on the algorithms being used for the coordinate conversion. It also introduces the resource from which the data that drives the app comes from. Chapter 3 reviews the Android platform, as well as some performance optimization considerations that are relevant to the implementation of the GNSS Tracking app. Chapter 4 outlines the requirements for the GNSS tracking app, including the basic user expectations of an Android app. The features and functionality of the app will also be covered. Chapter 5 presents the implementation details of the app, including the software architecture, the user interface, and the database. Chapter 6 presents the evaluation of the thesis research. It starts off with an explanation of the benchmark setup, continues with a presentation of the benchmarked results, and is followed by a discussion and analysis of the results. The chapter concludes with a review of the failures and limitations of the implemented product. Finally, Chapter 7 draws the conclusions on the performance and usability of the GNSS tracking app, including the algorithm analysis. Some possible future enhancements will be presented as well. Lastly, the Appendices contain the Java code segments detailed in this thesis. Appendix I contains the code for the conversion methods, Appendix 2 details the open-sourced packages used in writing the application, Appendix 3 describes the files needed to implement the Google Maps API on Android, and Appendix 4 contains the code for the data type precision tests.

1.3 Research Questions

On top of algorithm and performance analysis, the goal of this thesis is to evaluate the capabilities of the Android mobile device to support the requirements of a satellite tracking application. The question is whether the existing Android platform can be used

to build a GNSS tracking app with a historic data time slider, fulfilling all the user and technical requirements in a way that would be useful to companies such as Here.com.

This thesis will address the following research questions:

1 How does the GNSS tracking app help mapping and location organizations such as Here.com?
2 How does the GNSS tracking app contribute to the field of GPS technology?
3 How is the GNSS tracking app affected in terms of resource usage, performance, and usability?
4 What new knowledge was derived from the coordinate conversion algorithm benchmarks?
5 What are the critical limitations of running the GNSS tracking app on a mobile device?

1.4 Research Design and Methods

The research for this thesis started with a study of GNSS concepts and recognizing the problem statement presented by Here.com. The literature review for this thesis consists of studying conversion methods between two different coordinate systems. Four conversion algorithms were selected for study. In addition, a study was conducted on the Android architecture and Android programming concepts, with focus on performance optimization.

To prepare for the experimentation, the Android development environment was set up and the relevant API was studied for its feasibility to support the requirements of a satellite tracking application. Similar apps were also downloaded and tested. The two areas for experimentation include the app development and the coordinate conversion algorithm analysis. A prototype for the GNSS Tracking app was developed. Testing was conducted on the prototype, and areas for improvements were identified. Four coordinate conversion algorithms were implemented in Java, and then benchmarked for accuracy and performance. In addition, several areas of the app implementation were benchmarked and studied for possible performance improvements.

The performance of the app was most likely to be affected where there is interaction with large data sets. The amount of data needed for the app depends on how far back in history the user would like to view the satellite location path. Therefore, a part of the research for this thesis includes investigating areas where the app interacts with the data, and whether the app performance holds up in these areas. Examples of such areas include downloading data from the third party website, loading the data on startup, and operating the time-lapse slider.

User experience is taken into account when deciding the optimal amount of data that can be handled by the app. This involves balancing the response time of the app when it performs time-intensive calculations, with additional factors such as data accuracy and the maximum amount of data that can be loaded before the app crashes. For the usability benchmarks, experimentations and measurements were conducted for execution time, memory consumption, and data usage.

Based on the benchmarking results, improvements were made to the app and re-tested. Lastly, the final product was evaluated to validate the proposed solution.

2 GNSS Concepts

This chapter presents the background research conducted for the GNSS concepts used in this thesis. It introduces some basic concepts for GNSS coordinate conversion calculations to lay the foundation for the implementations discussed in the later part of this thesis.

2.1 Background

GNSSs are navigation satellites systems that provide location information on Earth, usually through GPS receivers. There are several ways to determine the current location of a device with a GPS receiver. One method is to determine which cell towers or Wi-Fi networks the user is connected to, and determine the approximate location of the user from there. This is the quickest method as this position is available immediately to the device [HowStuffWorks, 2005].

Another method is for devices with GPS receivers to receive data directly from the GPS satellites in orbit [HowStuffWorks, 2005]. This method is more accurate than using cell towers and Wi-Fi networks as it can determine the exact position of a user by using a process called trilateration. Trilateration uses the distance between the data from at least four GPS satellites and the receiver on the device to determine where the distances intersect. The point where the distances intersect is the location of the user. [Wang et al., 2008] Therefore, to help accurately determine a user's location with GPS satellites, it is necessary to know exactly where the satellites are in space [Kaplan, 1996]. For companies such as Here.com who are involved in location-based services, it would help to be able to visualize the path and locations of the satellites with respect to the user's location, and predict where the satellites will be at a given moment in time. Having better accessibility to satellite location data will help to create better tools and services in the future [Steiniger et al., n.d.].

GPS receivers use a combination of both GPS and GLONASS satellites for improved coverage and accuracy. The GPS navigation system currently has 31 active satellites in orbit around Earth, inclined at 55 degrees to the equator. These satellites orbit at 26,560 km from the centre of the Earth, and make two orbits in a sidereal day. The orbits are designed so that there are always 6 satellites in view, from most places on Earth [Tsui, 2000]. To see which GPS satellites are in view in a given position, a sky plot can be used.

There are many sources to retrieve historic data for past satellite location. One such source is the International GNSS Service (IGS) website [IGS, n.d.]. There are several navigation satellite systems orbiting Earth, and these are discussed in the next

section. Currently, the IGS only provides data for two GNSS, GPS and the Russian GLONASS [IGS, n.d.].

2.2 Navigation Satellite Systems

Currently, the United States NAVSTAR GPS and the Russian GLONASS are the most widely used globally operational GNSSs. These are the two GNSSs that will be represented in the GNSS Tracking app. There are other navigation satellite systems currently in orbit, such as Galileo, which is the GNSS that is currently being created by the European Union (EU) and the European Space Agency (ESA) [Astronautix.com, n.d.]. The BeiDou Navigation Satellite System (BDS) is a limited Chinese test satellite navigation system that has been operating since 2000 [Hegarty and Chatre, 2008].

In addition to GNSS systems, several countries have implemented their own satellite-based augmentation system (SBAS) which compensates for certain disadvantages of currently operational GNSS in terms of accuracy, integrity, continuity and availability [EGNOS, 2015]. SBAS only includes geostationary satellites. One example of SBAS is the Quasi-Zenith Satellite System (QZSS), which is a proposed three-satellite regional time transfer system for GPS that would be receivable within Japan [JAXA, 2010]. There are also Wide Area Augmentation System (WAAS) that is developed by the Federal Aviation Administration (FAA) to improve the accuracy, integrity, and availability of GPS [FAA, 2015]. WAAS consists of multiple geosynchronous communication satellites. As of June 2011, it consists of three commercial GEO satellites, namely the Inmarsat-4 F3, Telesat's Anik F1R, and Intelsat's Galaxy 15 [Navipedia, 2014].

2.3 Coordinate Systems

In geometry, three dimensional spaces are represented by the xyz-coordinate system. Similarly, a coordinate reference system (CRS) is a three-dimensional system for representing locations on the surface of the Earth. CRS is one of several coordinate systems in existence. The reason for this variety is because the Earth can be represented in a spherical model or as a flattened rectangular map. In addition, there are different purposes to location information, which requires different parameters and methods of calculations to accomplish [USGS, 2013].

There are three main criteria for defining coordinate systems. First of all, a coordinate system is defined by its measurement framework. There are two different measurement frameworks used in this thesis, the geographic coordinate system and the

Cartesian coordinate system. The geographical coordinate system, also called the geodetic coordinate system, uses spherical coordinates which are measured from the centre of the Earth. The geographic coordinates are given in longitude, latitude, and height (LLH) parameters. The Cartesian coordinate system, also called the geocentric coordinate system, uses coordinates in x, y, and z parameters. Other coordinate systems includes the Universal Transverse Mercator (UTM), which divides the world into 60 North-South zones; and the State Plane coordinate system, which is only used in the United States to divide the country into zones [USGS, 2013].

The second defining criterion of a coordinate system is its unit of measurement, which is decimal degrees for the geodetic coordinate system and in feet or meters for the Cartesian coordinate system [USGS, 2013].

Lastly, a coordinate system needs a reference datum, which details a list of known positions on Earth. Currently, the World Geodetic System (WGS 84) is the only world referencing system in place, and the default standard datum for coordinates stored in GPS units for personal and commercial use [USGS, 2013].

The data provided by IGS contains historic satellite locations in the Cartesian coordinate system, and the Google Map API plots locations in the geodetic coordinate system. Therefore, conversion for the data from the Cartesian coordinate system to the geodetic coordinate system is needed to be implemented.

2.3.1 Geodetic Coordinate System (Latitude, Longitude, Altitude)

As mentioned above, geographical or geodetic coordinates are spherical coordinates. Lines of latitudes run parallel to each other and to the equator of the Earth. Longitudes are lines that run from the North Pole to the South Pole. Latitude and longitude are measured in degrees, and altitude is measured in meters.

The latitude of the Earth is from -90 to 90 degrees with the equator at 0 degree. The longitude is from -180 to 180 degrees with the Greenwich meridian at 0 degree. The altitude is the height above the surface of the Earth [USGS, 2013].

From here on, the LLH coordinate system will be referred to as the “geodetic” coordinate system.

2.3.2 Cartesian Coordinate System (XYZ)

The Cartesian (or geocentric) coordinate system used in GPS is also called the Earth-Centered, Earth-Fixed (ECEF). ECEF uses three-dimensional XYZ coordinates (in meters) to describe the location of a GPS user or satellite. The z-axis is the rotational axis. It pierces the Earth at the poles and its positive direction is towards the North Pole.

The Earth rotates in the positive direction around this z-axis: when you lay your right hand on the globe such that the thumb points to the North Pole, then the Earth rotates in the direction of the fingers ("the rule of the right hand"). The x-axis points towards the intersection of the prime meridian with the equator. The y-axis is in the equator plane, at right angles to the x-axis [Dana, 2015].

From here on, the XYZ coordinate system will be referred to as the “Cartesian” coordinate system.

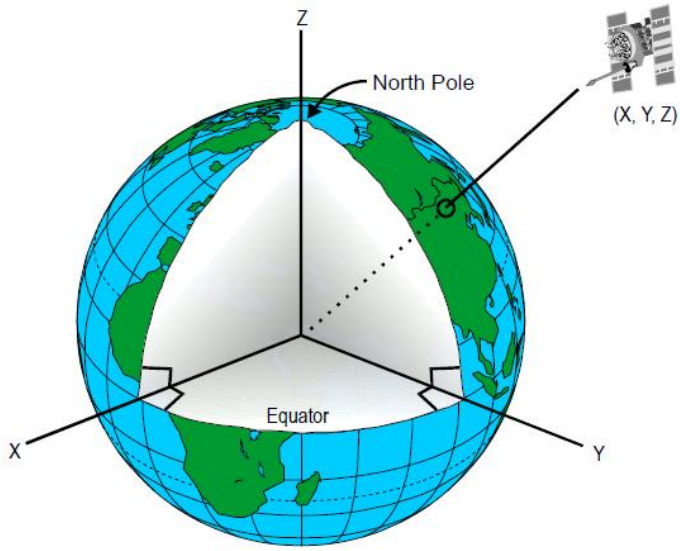


Figure 2.3.2.1 This is an image representing a satellite using the Cartesian coordinate system [FAA, 2003].

2.3.3 World Geodetic System (WGS-84)

A Geodetic datum or geodetic system is a coordinate system, and a set of reference points, used to translate positions on maps to their real locations on the Earth. Many of these datums are locally defined and apply only to maps in specific areas. For example, the North American Datum 1927 (NAD 27) is an example of a local geodetic system. Its reference surface is the Clarke 1866 ellipsoid; its origin is Meades Ranch in Kansas; and its area of applicability is North America. An example of a system that is applicable worldwide is the World Geodetic System 1984 (WGS 84). Its origin is at the centre of the Earth [Dana, 2015].

The GNSS Tracking app uses the Google Maps API to display the world map. As the internal coordinate system of Google Earth uses the geodetic coordinate system on the WGS-84 datum [Google, 2015a], parameters from the WGS-84 datum will be used for calculating satellite positions. For example, it is necessary to have parameters for the major (equatorial) radius (*a*) of Earth's ellipsoid at the equator, flattening (*f*), and the polar semi-minor axis (*b*).

$$
\begin{aligned}
a &= 6378137 \text{ m} \\
b &= a(1-f) = 6356752.31424518 \text{ m} \\
f &= \frac{1}{298.257223563} \\
e^2 &= \frac{a^2-b^2}{a^2} \\
\varepsilon^2 &= \frac{a^2-b^2}{b^2}
\end{aligned}
$$

Figure 2.3.3.1 WGS84 parameters for Earth's ellipsoid [NGA, 1984].

2.4 Cartesian to Geodetic Coordinates Conversion Methods

The conversion from geodetic to Cartesian coordinates is a straightforward task. However, converting from Cartesian to geodetic is a complicated problem [Heiskanen and Moritz, 1967].

If the Earth is an ideal sphere, the calculation of geodetic coordinates would be quite straightforward. The calculation of the position of an object on the Earth's surface is simply the calculation of a point on the surface of a sphere using three-dimensional coordinates [Dana, 2015]. The value of longitude (λ) and latitude (φ) can be computed using trigonometric ratios [Kaplan, 1996]. In addition, the altitude of the object off the Earth's surface would be to subtract the position of the object (r) from the theoretical radius of a spherical Earth (r_e).

Distance from the Earth's centre	$r = \sqrt{x^2 + y^2 + z^2}$
longitude	$\lambda = \tan^{-1}\left\{\frac{y}{x}\right\}$
latitude	$\varphi = \tan^{-1}\left(\frac{z}{\sqrt{x^2+y^2}}\right)$
altitude	$h = r - r_e$

The relationship between Cartesian and geodetic coordinates can be simplified in Figure 2.4.1.

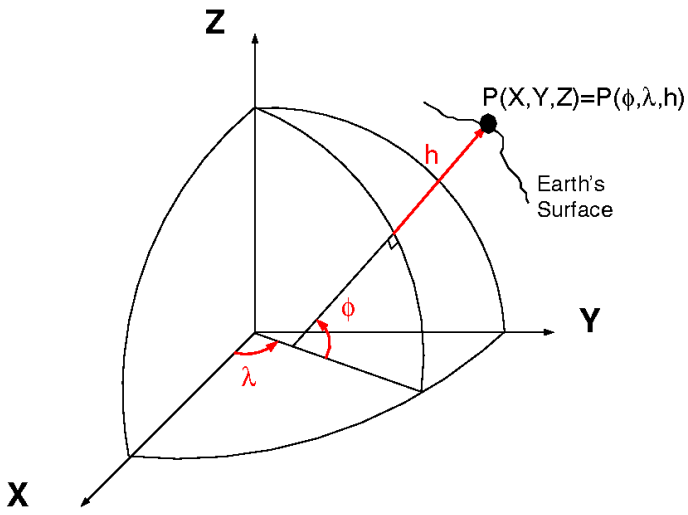


Figure 2.4.1 This is an image representing the difference between the xyz-coordinate system and the LLH-coordinate system [NGS, 2003].

However, Earth is flattened at the poles and its circumference is bigger at the equator. Therefore, an ellipsoid is chosen to approximate its shape and the aforementioned equations must be modified. The calculation for the value of longitude (λ) is quite straightforward. The initial value is computed the same as that of an ideal spherical Earth, except the final value needs to be compensated depending on whether the object is on the East or West of the Prime Meridian [Kaplan, 1996], as such:

$$\lambda = \begin{cases} arctan\left(\frac{y}{x}\right), & x \geq 0 \\ 180° + arctan\left(\frac{y}{x}\right), & x < 0 \; and \; y \geq 0 \\ -180° + arctan\left(\frac{y}{x}\right), & x < 0 \; and \; y < 0 \end{cases}$$

The calculation of geodetic latitude (φ), however, is more complicated. In fact, there have been studies of more than 70 papers spanning over five decades describing solutions or comparisons of solutions [Featherstone and Claessens, 2008]. The reason for the complication of accurate latitude calculation is due to the fact that the radius of curvature (*N*) is used in the calculation, but the formula for *N* contains the latitude itself [Burtch, 2006]. Therefore, to calculate the latitude using *N*, it is necessary to first

determine an initial estimate of φ, usually by using the formula for the ideal spherical Earth, as such:

$$\varphi_1 = \tan^{-1}\left(\frac{z}{\sqrt{x^2 + y^2}}\right)$$

The following subsections details four algorithms for calculating latitude that will be benchmarked for performance on the Android platform. The selection of which conversion method to implement covers three criteria:

1. Is it easy to implement as a programming algorithm?
2. Is it accurate?
3. Is it performance efficient? The efficiency of an algorithm depends on two factors, (a) the number square roots, cube roots and trigonometric functions requiring evaluation and (b) the number of iterations required for the indirect methods [Gerdan and Deakin, 1999].

2.4.1 Closed Solution Method by Kaplan [Kaplan, 1996].

The following 15 step procedure summarised by Kaplan is a method that is highly accurate [Zhu, 1994]. It is selected for research for this thesis for the fact that it uses only one trigonometric function. It will be determined if this will affect its efficiency in terms of performance and memory usage.

$$r = \sqrt{X^2 + Y^2}$$
$$E^2 = a^2 - b^2$$
$$F = 54b^2Z^2$$
$$G = r^2 + (1 - e^2)Z^2 - e^2E^2$$
$$C = \frac{e^4Fr^2}{G^3}$$
$$S = \sqrt[3]{1 + C + \sqrt{C^2 + 2C}}$$
$$P = \frac{F}{3(S + \frac{1}{S} + 1)^2G^2}$$
$$Q = \sqrt{1 + 2e^4P}$$
$$r_0 = \frac{-(Pe^2r)}{1 + Q} + \sqrt{\frac{1}{2}a^2\left(1 + \frac{1}{Q}\right) - \frac{P(1 - e^2)Z^2}{Q(1 + Q)} - \frac{1}{2}\mathrm{Pr}^2}$$
$$U = \sqrt{(r - e^2r_0)^2 + (1 - e^2)Z^2}$$
$$V = \sqrt{(r - e^2r_0)^2 + (1 - e^2)Z^2}$$
$$Z_0 = \frac{b^2Z}{aV}$$

Altitude, $h = U(1 - \frac{b^2}{aV})$

Latitude, $\varphi = \arctan\left(\frac{Z + e'^2 Z_0}{r}\right)$

2.4.2 Hirvonen and Moritz's method [Burtch, 2006]

This method omits the geodetic height from the iterative calculation of latitude, φ. The radius of curvature in the prime vertical and geodetic latitude are then continually refined until the maximum error between successive iterations of height calculations is less than the acceptable precision [Frame and Coffey, 2014].

distance from the polar axis to the point $p = \sqrt{x^2 + y^2}$

initial estimate of geodetic latitude, φ $\varphi_1 = \tan^{-1}\left\{\frac{z}{p}\left(1 + \frac{e^2}{1-e^2}\right)\right\}$

iterate φ_1 until the difference in φ is insignificant.

radius of curvature in the prime vertical $N = \frac{a}{\sqrt{1-e^2 \sin^2 \varphi_1}}$

refined latitude, φ $\varphi = \tan^{-1}\left\{\frac{z}{p}\left(1 + \frac{e^2 N \sin(\varphi_1)}{z}\right)\right\}$

assign to φ and iterate $\varphi_1 = \varphi$

geodetic height $h = \frac{p}{\cos \varphi_1} - N$

2.4.3 Torge's method [Torge, 2001]

Torge's equation differs slightly from Hirvonen and Moritz in the equation to refine latitude, φ_1. Specifically, it calculates the geodetic height first and uses it in the equation to refine the geodetic latitude.

distance from the polar axis to the point $p = \sqrt{x^2 + y^2}$

initial estimate of geodetic latitude, φ $\varphi_1 = \tan^{-1}\left\{\frac{z}{p}\left(\frac{1}{1-e^2}\right)\right\}$

iterate φ_1 until the difference in φ is insignificant.

radius of curvature in the prime vertical $N = \frac{a}{\sqrt{1-e^2 \sin^2 \varphi_1}}$

geodetic height $h = \frac{p}{\cos \varphi_1} - N$

refined latitude, φ_1 $\varphi_1 = \tan^{-1}\left\{\frac{z}{p}\left(\frac{1}{1 - e^2 \frac{N}{N+h}}\right)\right\}$

assign to φ and iterate $\varphi = \varphi_1$

2.4.4 Bowring's method [Bowring, 1985]

Bowring developed a rapidly converging iterative method based on Newton's method for computing tan φ [Bowring, 1976]. This method uses three calculations of tangent for evaluating the latitude. However, the computation of tan φ is quite expensive, performance-wise [Burtch, 2006]. Bowring refined his algorithm in 1985 and this resulted in an improvement in the initial estimate of the parametric latitude, β. In addition, only one calculation involving tangents is required, with the other tangent functions being able to be substituted out without it being necessary to be evaluated for result. This improved the performance of Bowring's method. On top of that, Bowring's refined algorithm also benefit from being highly accurate such that only one iteration of the method is required [Bowring, 1985]. This conclusion is verified as well by the experimentations conducted by this thesis.

1st eccentricity of ellipsoid	$e^2 = \frac{a^2-b^2}{a^2}$
2nd eccentricity of ellipsoid	$\varepsilon^2 = \frac{a^2-b^2}{b^2}$
distance from minor axis	$p = \sqrt{x^2 + y^2}$
polar radius	$R = \sqrt{p^2 + z^2}$
initial estimate of parametric latitude, β	$tan\beta = \frac{bz}{ap}\left(1 + \varepsilon^2 \frac{b}{R}\right)$
iterate until the difference in φ is insignificant:	
geodetic latitude	$tan\varphi = \frac{z+\varepsilon^2 b\, sin^3\beta}{p-e^2 a \cos^3\beta}$
refine parametric latitude, β_1	$tan\beta_1 = \frac{b}{a} tan\varphi$
assign to β and iterate	$\beta = \beta_1$
length of normal terminated by minor axis	$v = a\sqrt{1 - e^2 \sin^2 \varphi}$
height above ellipsoid	$h = p\, cos\varphi + z \sin\varphi - \frac{a^2}{v}$

2.5 Satellite Location Data from the International GPS Service (IGS)

There are several methods of obtaining satellite data. One of which is easily provided by the Google API via the `android.location.GpsStatus` class [Android, 2015b]. The API provides a list of satellites that the device is currently tracking. The API also provides information on the satellites' azimuth, elevation, pseudo-random number (PRN), signal-to-noise ratio (SNR), and whether the satellite was used by the GPS engine to calculate the most recent GPS fix. This method is commonly used when real time information on the satellites is required.

If historic data on the satellite path is required, one can retrieve it via a different method from a third party source. The GNSS Tracking app uses this method to display historic satellite data in its time slider view. In this thesis, the third party source is the International GPS Service (IGS) [IGS, 2015].

The IGS is a voluntary collaboration of more than 200 contributing organizations in more than 80 countries. The IGS global tracking network operates more than 300 permanent, continuously-operating GPS stations, and has provided GPS data to the public since 1994 [IGS, 2009]. The GPS data is available on their FTP site at `ftp://ftp.igs.org`. The IGS FTP site offers post processed accurate satellite locations that can be downloaded for any particular day. The files are in .sp3 format, and come compressed as .Z files. The coordinates are logged in WGS-84 ECEF format at 15-minute intervals. There is a lot of information compressed into the .sp3 file, including data that is not used by the GNSS Tracking app. An example of a .sp3 file is show in Figure 2.5.1.

```
#cP2015  5  4 18  0  0.00000000     192 ORBIT IGb08 HLM  IGS
## 1843 151200.00000000   900.00000000 57146 0.7500000000000
+   55   G01G02G03G04G05G06G07G09G10G11G12G13G14G15G16G17G18
+        G19G20G21G22G23G24G25G26G27G28G29G30G31G32R01R02R03
+        R04R05R06R07R08R09R10R11R12R13R14R15R16R17R18R19R20
+        R21R22R23R24  0  0  0  0  0  0  0  0  0  0  0  0  0
+          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
++         4  3  3  4  3  4  4  4  3  4  3  4  4  4  4  3  4
++         4  4  4  4  4  4  4  6  3  3  3  4  3  3  5  4  4
++         5  5  4  4  5  4  4  5  5  4  6  5  4  4  4  5  5
++         5  4  4  5  0  0  0  0  0  0  0  0  0  0  0  0  0
++         0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
%c M  cc GPS ccc cccc cccc cccc cccc ccccc ccccc ccccc ccccc
%c cc cc ccc ccc cccc cccc cccc cccc ccccc ccccc ccccc ccccc
%f  1.2500000  1.025000000  0.00000000000  0.000000000000000
%f  0.0000000  0.000000000  0.00000000000  0.000000000000000
%i    0    0    0    0      0      0      0      0         0
%i    0    0    0    0      0      0      0      0         0
/* ULTRA ORBIT COMBINATION 18432_18 (57147.750) FROM:
/* cou emu esu gfu gou ngu siu usu coR emR esR gfR
/* REFERENCED TO emu CLOCK AND TO WEIGHTED MEAN POLE:
/* PCV:IGS08_1842 OL/AL:FES2004  NONE     Y  ORB:CMB CLK:CMB
*  2015  5  4 18  0 0.00000000
PG01 -12554.401080 -13363.117282 -19331.269697     -5.992515  8  7  8 196
PG02 -10483.144700  14820.284460  19608.276564    563.795210  6 10  7 203
PG03 -13630.001532 -22771.576700    -17.680455    317.558013  6  7  8 173
PG04  -2144.670622 -15747.030589 -21613.536352    -13.493776  5  8  7 204
PG05   -958.860812  20889.180252  16215.441169   -245.687516  6  7  7 195
PG06 -23848.200477   6979.820107   9372.777413    152.371362  8 10  7 183
PG07 -23896.927135  -6971.938531   9873.671261    459.883519  2 11  8 177
PG09 -12977.246654  -8079.504325  21710.872588   -150.034614  7  5  9 203
PG10  15806.825661   2096.252540  20805.425898    173.546904  7 10    196
```

Figure 2.5.1 A sample of the .sp3 file available from IGS. The name of this file is igu18425_00.sp3 [IGS, 2015].

The information that is applicable to the GNSS Tracking app begins on line 23, which contains the date of the data. Following that, on line 24, is the satellite vehicle id followed by its x-coordinate, y-coordinate, and z-coordinate. This continues for all the vehicles for the satellite system, from line 24 to line 78. This format of data continues 192 times in the same file, containing two days' worth of data in 15-minute intervals.

3 Android Concepts

This chapter presents the background research conducted on the Android platform as it pertains to the implementation of the GNSS Tracking app. It also discusses performance and memory optimization for the GNSS Tracking app.

3.1 Android Platform Overview

The operation system of the Android platform is based on the Linux kernel. System services for Android, such as security, memory management, and process management are controlled by Linux. The next layer that sits on top of the Linux kernel contains the middleware, libraries, and APIs written in C. Next, the application software running on the application framework includes Java-compatible libraries based on Apache Harmony [Brady, 2008].

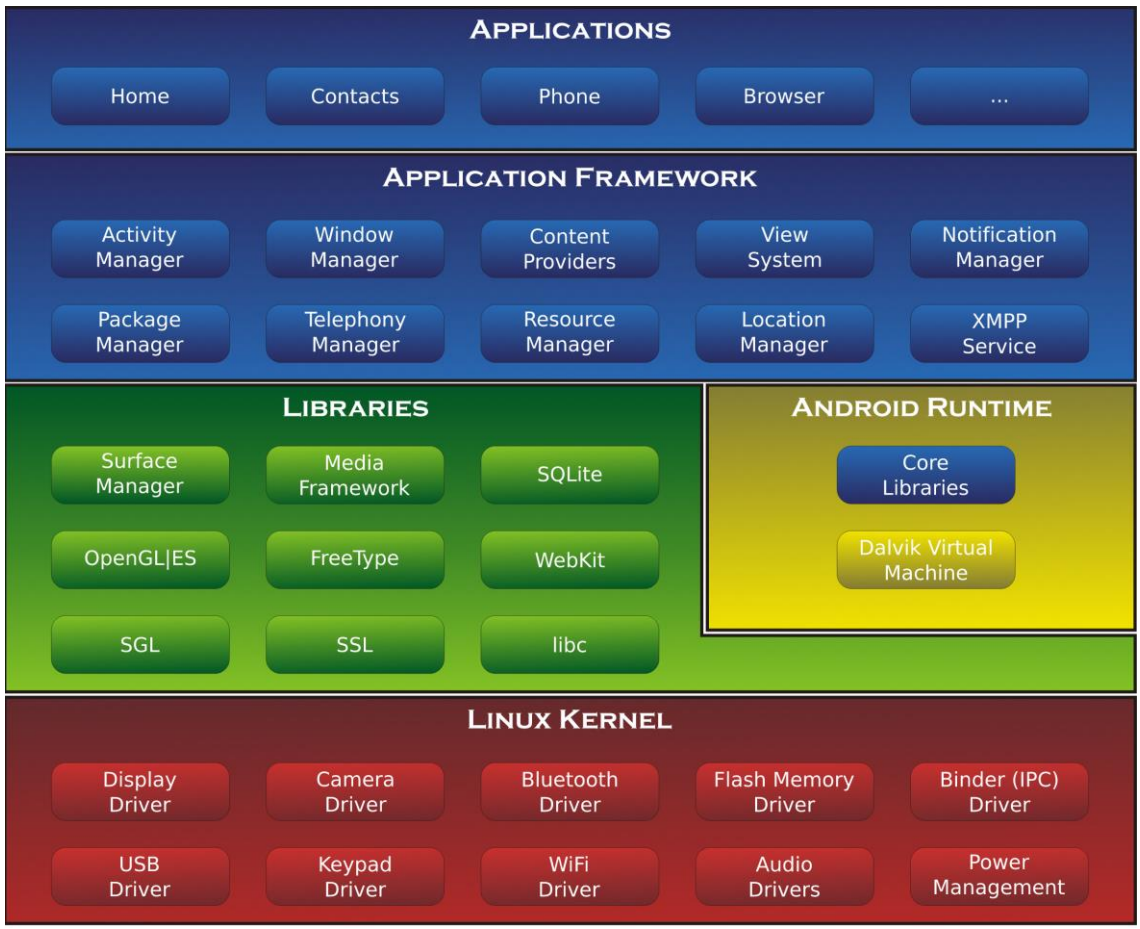
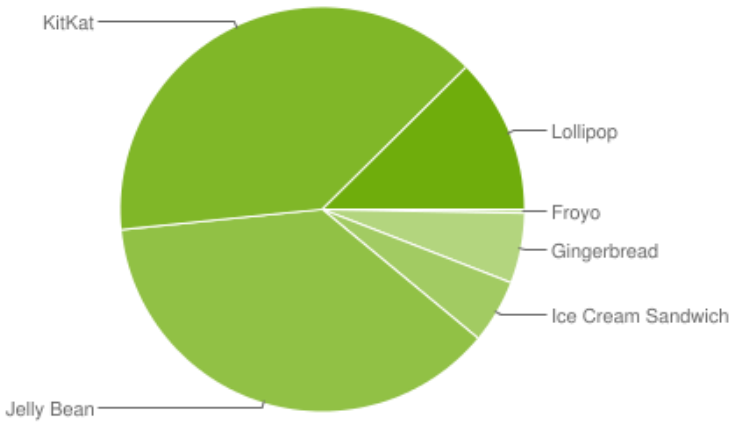


Figure 3.1.1 The system architecture of Android [Brady, 2008].

Android uses the Dalvik process virtual machine to execute apps on top of its main operating system (OS). This allows each app to run in its own virtual machine space, without affecting the execution of other apps. In Android 4.4, a new experimental virtual machine called Android Runtime (ART) was introduced by Google as the new runtime environment. ART uses a new Ahead-Of-Time (AOT) concept contrary to Dalvik's Just-In-Time (JIT) compiler. AOT fully compiles apps at the moment of installation. Dalvik will compile apps on demand and store it in the RAM until the RAM runs out, at which the compiled app will be unloaded and need to be recompiled again the next time it is launched. Although first-time app installs takes longer in ART, but subsequent usage is faster and performs better than Dalvik on mobile devices. In Android 5.0, the ART runtime replaces Dalvik as the platform default [Android, 2015g]. The device used for testing in this thesis was the Samsung Galaxy S4 (GS4) running Android 4.2.2 (Jelly Bean). ART is not in the GS4 Developer Options. Therefore, the benchmarking conducted for this thesis is on the Dalvik VM.

One of the goals involved in building a mobile native app is to have it working for as many versions of its OS as possible, without compromising features or functionality. There are currently ten versions of Android, as seen in Figure 3.1.2 below. The most widely distributed version is Jellybean and Kit Kat (Android 4.1 – 4.4), with the latest version being Lollipop (Android 5.x) [Android, 2015e]. The GNSS Tracking app will be developed to support Android 4.2 and above, which will cover 73.76% of the distributed Android versions.



Android Version	Android OS	API	Distribution
2.2	Froyo	8	0.30%
2.3.3 - 2.3.7	Gingerbread	10	5.60%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.10%
4.1x	Jelly Bean	16	14.70%
4.2x		17	17.50%
4.3		18	5.20%
4.4	KitKat	19	39.20%
5	Lollipop	21	11.06%
5.1		22	0.80%

Figure 3.1.2 Breakdown of active devices running different versions of the Android platform, as of June 1, 2015 [Android, 2015e].

Android apps are written using the Android software development kit (SDK). Developers have access to the Android API, including the Google Maps API for displaying maps, which eases the implementation of the world map. There is also the `android.location.GpsStatus` class for retrieving current satellite location information, which eases the implementation of the sky plot without the need for cumbersome data retrieval from third party resources. The Android API also contains built-in features for adding time lapse sliders for the user interface and asynchronous schedulers for the nightly data download process.

3.2 Optimizing for Android

Mobile applications are still considered software applications, and the standard software development good practices should still apply. There are some slight differences in development practices, mainly due to the smaller scale of mobile application projects which makes it easier to manage, debug, and optimize, but also promote a less formal development process [Agrawal and Wasserman, 2010]. Part of good software development practice includes optimizing app performance, usability, reliability, and ease of maintenance. The literature review for this thesis include studying methods to optimize mobile apps for Android and implementing these methods to improve the app. Listed below are some optimization considerations for developing Android apps.

3.2.1 Performance Optimization

Ideally, the best possible performance for an app on the Android platform would be to implement it in pure C using the Native Development Kit (NDK) of Android, compiled to binary code and executed directly under the Linux kernel [Lee and Jeon, 2010], but this does not guarantee that the app would be more efficient. In addition, this would defeat the goal of having the app be easily customizable. The NDK documentation specifically states that "using native code on Android generally does not result in a noticeable performance improvement, but it always increases your app complexity" [Android, 2015d].

Without modifying the native code, there are still many other options for optimizing for Android. Android has documentation for developers on performance optimization tips. It lists several recommendations for good programming practices, such as to avoid creating unnecessary objects and to avoid allocating memory if possible. Some data access recommendations include making methods and constants static as it is 15% to 20% faster, to use direct field access instead of getters and setters as it is three times as fast, and to use integers instead of floating point data types as it is twice as fast [Android, 2015f]. All of the above recommendations are feasible to be implemented in the GNSS Tracking app. However, on the last point of using integers instead of floating point data types, it is known that integers are not practical for scientific calculations as it does not allow decimal values. The `float` and `double` primitive types in Java are commonly used to represent numerical values in scientific applications to conduct arithmetic operations, both of which are floating point representations of numbers. On that regard, it is commonly known that rounding errors are inherent in floating-point computation. Java has a `BigDecimal` class which will handle very large numbers and very small numbers, but on top of being overly-verbose in its method calls, the `BigDecimal` class suffers in performance due to its high memory consumption. Another workaround to the precision issue would be to use another programming language on the Android platform, such as Scala [Scala, 2015]. However, the downside of using Scala on Android is that it suffers in performance and over consumption of memory [Denti and Nurminen, 2013]. Other possible alternatives include Scientific Python [SciPy, 2015], the GNU Multiple Precision Library [GNU, 2014], or the GNU MFPR Library [GNU, 2015]. While these alternatives were not explored in the research for this thesis, it could be investigated as a future work to this thesis.

Conserving memory is of utmost importance in improving performance efficiency, especially more so on a mobile device where physical memory is often limited. This is because when the device is running low on memory, it will perform garbage collection. The garbage collection process in Android will create pauses in the

user experience, making them wait for the process to be completed before the app can continue functioning. Therefore, it is advisable to avoid introducing memory leaks and to release object references where possible. Areas where this is possible include releasing memory when the user interface becomes hidden; optimizing image files for the appropriate resolution size, and using optimized data containers.

Some preventive programming practices involves being aware of the memory overhead of the program, and being careful about using external libraries. The app will be benchmarked to monitor memory consumption. The GNSS Tracking app uses code fragments from two open-sourced resources. One is the code module from the `compress-j2me` project, which is a compression module used to extract the files from the IGS site. The data from the IGS FTP site is offered in .sp3 files, and comes compressed as .Z files. Files with .Z extensions are compressed using the UNIX compression utility based on the Lempel-Ziv-Thomas (LZC) compression method [Frysinger, 2015]. Another code module used is the YGPS Satellites application [Yunnanexplorer, n.d.] for Android, which shows the location and signal strength of the GPS satellites on a sky plot, as this was a feature requested by the client. Both of these code modules were examined for security leaks, and monitored during the app execution for memory leaks.

3.2.2 User Experience Optimization

Performance is only one of the many considerations of mobile application development. Other important requirements include the user experience and reliability of using the application, and the human resource needed to build and maintain the application [Wasserman, 2010].

For a mobile application, one of the key components of user experience is responsiveness. In Android, responsiveness of the app is monitored by the Activity Manager and Window Manager system services. Android will display an “Application Not Responding" (ANR) message when the application does not respond to a user input event within five seconds, or when a `BroadcastReceiver` has not complete execution within ten seconds [Android, 2015j]. Therefore, long-running tasks should not run in the User Interface (UI) thread, or the main thread, and should be deferred to a background worker thread or an asynchronous request. This includes tasks to download data from an external source, network or database operations that handles large amounts of data, or calculations that would consume large amounts of memory resource. This will ensure that the user does not have to wait too long for a response from the app.

One of the ways to optimize app responsiveness is to optimize the layout performance. This entails smooth scrolling interfaces with a minimum memory footprint. The app can reduce memory usage and speed up rendering by loading the

views only when they are needed [Android, 2015n]. The app can also store UI component views inside the tag field of the Layout using a `ViewHolder` object. This allows the app to access the views immediately without the need to look them up repeatedly [Android, 2015o]. The Android SDK provides an Hierarchy Viewer tool that detects bottlenecks in the app's layout performance while the app is running [Android, 2015p].

In addition to the immediate user experience of using the app, there is also the long-term or background user experience to consider. For example, having an app that drains battery life at a high rate or interferes with the operation of other apps would not affect the immediate user experience of interacting with the app, but would impact the long-term satisfaction of owning the app. If the app has to download large amounts of data that may require a high bandwidth, those tasks should be deferred to when the device is connected to Wi-Fi. A `BroadcastReceiver` can be implemented to listen for connectivity changes and to initiate a download only when the device is connected to Wi-Fi [Android, 2015k]. A `ConnectivityManager` can be used to check the type of internet connection that the device is connected to, whether it is Wi-Fi or cellular [Android, 2015l]. Updates that may drain the battery of the device should be deferred to when the device is connected to a wall charger. Power-intensive updates should be reduced when the battery is discharging. It should even stop updates completely when the battery power is running on low [Android, 2015m].

4 Requirements

This chapter details the requirements for the GNSS Tracking app. On top of the requirements provided by Here.com, it also details the basic requirements of a mobile app which makes the user experience reasonably pleasant. Additional features and functionality were also brainstormed and implemented to enhance usability of the app.

4.1 Hardware Requirements

There are a few hardware requirements that need to be met for using the GNSS Tracking app, mainly due to the client's request. The app requires at least Android OS 4.2.2. The app also requires that the device has touch-screen capabilities. Wi-Fi is an optional requirement, mainly to reduce data costs of downloading the historic data cache via the user's data plan. The sky plot does require that the device have a GPS receiver to work properly, but that is not required to use the world map time lapse slider which only displays cached historic data obtained from a third party source. Figure 4.1.1 summarizes the minimum hardware requirements for the app.

Android version:	Android OS 4.2.2
CPU:	1 GHz
GPU:	200 MHz
Memory:	512 MB RAM
Display:	4.0" 800x480 pixels (233 ppi)

Figure 4.1.1 Minimum hardware requirements.

There are no additional requirements for a camera, microphone, audio or any other hardware sensor.

4.2 Basic Requirements for Mobile Applications

The GNSS Tracking app should fulfil a minimum of the behaviour expected of a quality mobile app, as well as the basic requirements from the client. First of all, the basic features of a mobile device should be available. For example, a majority of mobile devices today utilizes the touch screen feature for navigation within the device.

Therefore, the app implementation should leverage the touch screen feature, and should not rely entirely on the device buttons. Next, the app should have a relatively intuitive navigation and user interface. At the very least, the app should not require that the user read a manual and spend more than a day learning how to use it. Following that, the user experience of interacting with the app should be sufficiently pleasant. This includes not making the user wait more than ten seconds for the app to respond to requests. If an activity is anticipated to take longer than ten seconds to complete, it should either be delegated to a background task or a progress loading screen should be displayed to the user [Nielson, 1994]. It also includes not having navigational dead ends to the app interaction that requires the user to force a re-start of the app in order to get back to the main screen, In addition, the app should not crash too often. Lastly, the app must not ruin the overall experience of using the mobile device. For example, the background tasks initiated by the app should not utilize more than the reasonable amount of memory of the device. It should not slow down the performance of other apps or interfere with the other background tasks outside of the app. It should not cause memory leaks, or interrupt basic mobile features such as alarms and incoming calls. In addition, the app should not be too costly to operate, in a way that it drains the user's mobile data plan. On top of that, the app should not drain battery life at a higher than normal rate.

4.3 Functional Requirements

The client, Here.com has provided a list of requirements for the GNSS Tracking app, as listed below.

App Settings: The client wants to be able to control settings of the app, such as to choose which satellite systems (GPS or GLONASS) to display.

ID	Requirements for App Settigs
F-1	User can choose which satellite systems are visualized on the UI
F-2	The user can set an arbitrary location as the reference location by inserting the WGS-84 coordinates (latitude and longitude in degrees/minutes/seconds)
F-3	The user can set an arbitrary time as the reference time by inserting the time in UTC+00 (year/month/day/hour/minute/second) and the UTC TimeZone. Future predictions limited to +2 days from the current time.

Backend requirements: The system will display the date and time on the app in UT and local time zones. The system will also have a schedule to fetch and parse the historic satellite location data for caching on the device storage.

ID	Requirements for Backend
F-4	The application uses the current location and time (UTC+TimeZone) of the device as the default location and reference time
F-5	The application will visualize on the UI the UTC-time and the UTC TimeZone
F-6	The application will update the constellation statuses (Skyplot, Worldmap) at the default update rate (i.e. UI refreshed e.g. at 1Hz)
F-7	The application will automatically download a new set of satellite ephemerides at the most convenient update rate to keep the satellite location and health information valid. Health information can be read/parsed from NANU/NAGU reports

World map: The app should be able to display a world map with the real-time location of the user, and the satellite locations as per the coordinate data from IGS. The user is able to view the changing location of the satellites on the world map via a time lapse slider. The user should also be allowed to move around or zoom in and out of the map.

ID	Requirements for World Map
F-8	User can freeze the UI by pausing the updates
F-9	The application will indicate the overall health of the satellites e.g. by color coding
F-10	The application will indicate the reason of the satellite health/unhealth status by a color/number/text or by allowing the user to reveal more status information by pointing/touching the satellite icon on the UI
F-11	User can zoom in/out the world map

Sky Plot: The app should be able to display a sky plot with the real-time location of the user.

ID	Requirements for Sky Plot
F-12	User can modify the reference time e.g. by a slider to see the changes in the Skyplot (fast-forward/fast-backward)
F-13	User can modify the reference time e.g. by a slider to see the changes in the Skyplot (fast-forward/fast-backward) in the "pause-mode". Future predictions limited to +2 days from the current time.
F-14	User can select and set the reference location for the Skyplot by touching the world map

In addition, brainstorming for this project has yielded a few extra requirements to enhance the user experience. For example, the app includes a user-defined setting to choose whether to only download data when connected to a Wi-Fi network. The user is

also able to set the desired time where the daily data download is to be scheduled, and the number of days' worth of data is to be cached.

4.4 Non-Functional Requirements

A non-functional requirement elaborates a performance characteristic of a system, and how a system should perform certain tasks. Some basic non-functional requirements will be covered in the app implementation.

Performance: This is the crux of the thesis work. Performance in several areas of the app were benchmarked and optimized, such as algorithm efficiency in coordinate conversion methods, retrieving data from external sources, and displaying satellite locations on the world map.

Capacity: It is a known problem that mobile devices have limited storage capacity. Part of the optimization procedure of this thesis work involves experimenting with the different amounts of data that can be downloaded and stored for the GNSS Tracking app while still maintaining optimal performance and usability. It is important for a mobile app that the amount of data required to operate the app is well within reasonable limits.

Availability: The areas where the app may suffer in availability would be with the third party resource used to retrieve the satellite data. The source is known to be unpredictable in terms of availability. This in turn affects the status of the data presented to the user. In addition, the data obtained is known to be occasionally inaccurate or corrupted.

Reliability: The app is designed to be operational without an internet connection, as the satellite data is cached on the built in SQLite database on the device. If the external data source for the historic satellite data is not available, there is an option for the user to redirect data download from other resources.

Recoverability: The app is designed to handle exceptions and errors gracefully. A study was done on possible errors that might occur. Those errors were simulated and error handling code was implemented in those areas.

Usability: The app uses design concepts recommended by Google. This is a move by Google to ensure consistency in the Android user experience. Examples include

an action bar on the top of the screen to provide hints to users on navigation, minimum font size for readability, as well as recognizable icons for ease of adopting the app [Android, 2015c].

5 Implementation

This chapter discusses the implementation details of the GNSS Tracking app. It discusses the tools used to create the app, the system architecture, key parts of the code base, and design of the user interface.

5.1 Tools and Technology

Android is an open sourced software tool, which means that the source code is open for customization and integration, and anyone can implement a tool or a library to integrate with the platform. Therefore, there are a wide variety of tools and technologies on the market to choose from to accomplish the desired solution. Here are the tools and technologies utilized for this thesis work.

Development environment:	Android platform
Programming language:	Java
Database:	SQLite
IDE:	Android Studio

Figure 5.1.1 Tools and technologies used in implementing the GNSS Tracking app.

5.2 Software Architecture

The GNSS Tracking app uses a standard Android project layout as its underlying architecture. This is illustrated in Figure 5.2.1. Firstly, the `AndroidManifest.xml` is an important file for any app in the Android platform. It describes the functionality and requirements to Android, such as the name of the app to be displayed on the device, the starting activity, or what services are allowed to be executed in the app [Android, 2015i].

The `MainActivity` class is the starting point of the GNSS Tracking app. As with any `Activity` class, it can communicate its requests to the `ContentProvider` (`android.content.ContentProvider`) to access data for the app, or to the `BroadcastReceiver` (`android.content.BroadcastReceiver`) to get ready for any changes in the app, or to `Services` that operates background tasks and jobs [Android, 2015h].

When the GNSS Tracking app loads, the `MainActivity` class will first determine if there is any satellite location data stored on the device. If there is data is found, the `MainActivity` class will ask the `LocationActivity` class to display a map of the Earth with the satellite locations plotted on the map. However, if the data is missing, the `MainActivity` class will load the the `RefreshActivity` class, which will prompt the user to download the data.

Downloading satellite data will take some time. This is because the app has to connect to a third party website to retrieve the data, uncompress it, convert the data into a format that can be used by the Google Maps API, and save the data into the built-in database on the device. Therefore, the app offers the user the option to decide the amount of data to download, and if the download should take place only when the device is connected to Wi-Fi. The download task extends the Android `Service` class (`android.app.Service`) for this purpose. Since a service does not actually run in a new thread or process from the `Activity`, it is necessary to create a class to ensure that the download task is executed asynchronously in the background. There is an option to use a `Thread/Handler` design or the built-in `AsyncTask` (`android.os.AsyncTask<Params, Progress, Result>`) class provided with the Android SDK [Android, 2015a]. The GNSS Tracking app chooses to use the `AsyncTask`.

The resource for the satellite location data is stored on the IGS website, which is a third party website. The GNSS Tracking app makes a GET request to the website using an `HttpURLConnection` (`java.net.HttpURLConnection`), and reads the data using `InputStream` (`java.io.InputStream`).

The data from the IGS website is offered in .sp3 files, and comes compressed as .Z files. Files with .Z extensions are compressed using the UNIX compression utility based on the Lempel-Ziv-Thomas (LZC) compression method [Frysinger, 2015]. Open sourced compression modules are used to extract the files, specifically, code modules from the `compress-j2me` project [GitHub, 2015b].

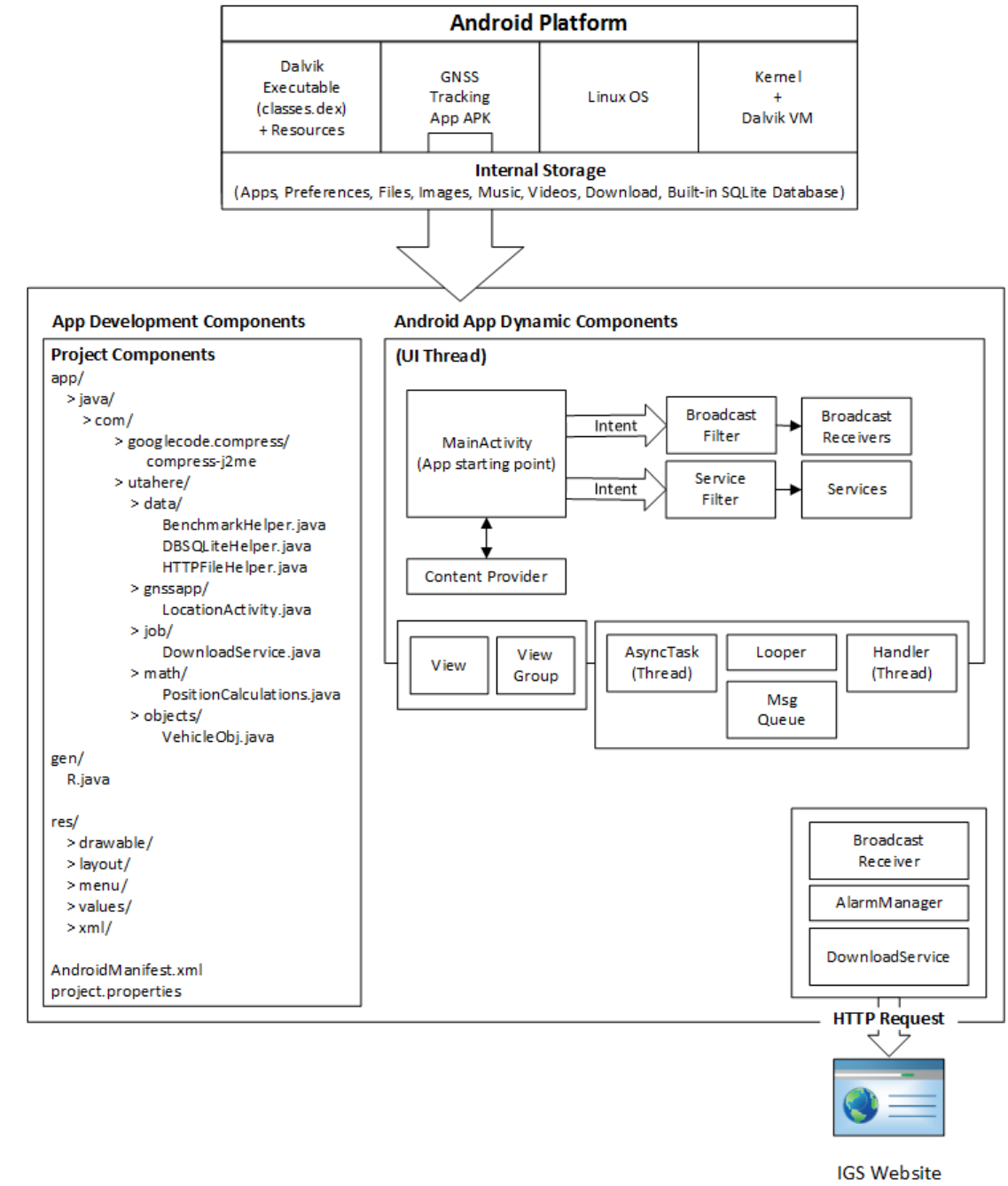
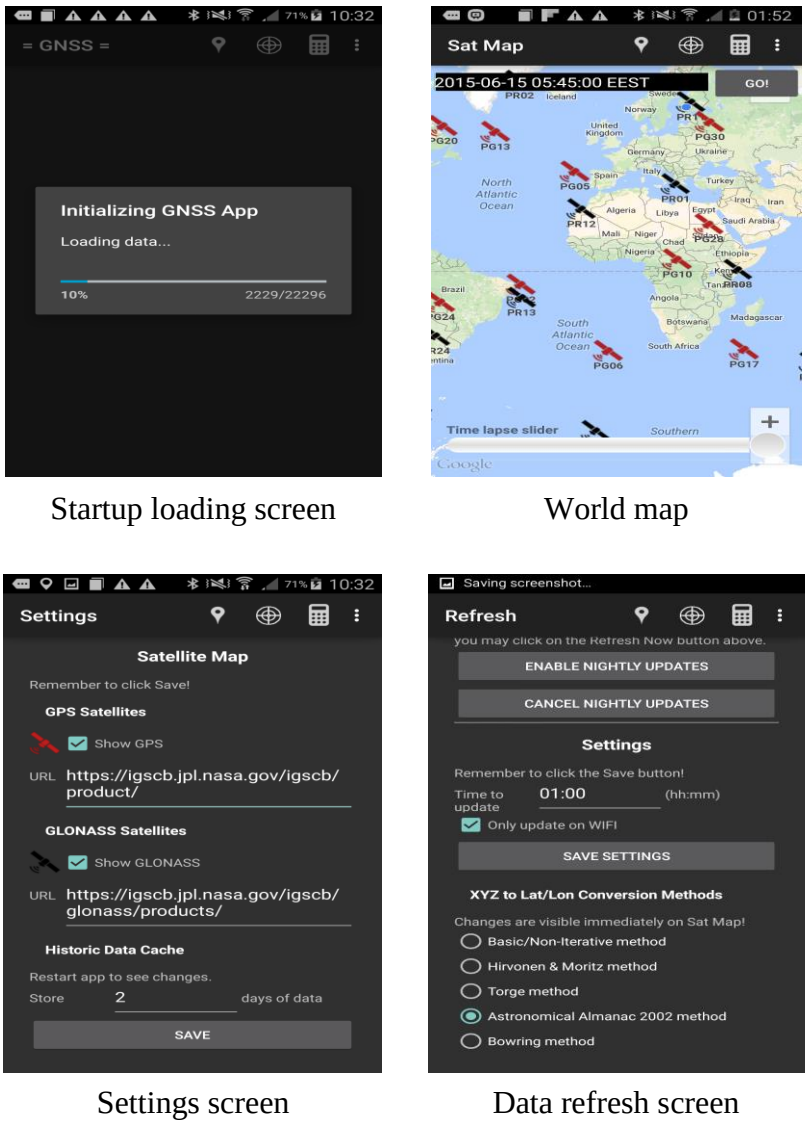


Figure 5.2.1 GNSS Tracking App System Architecture.

5.3 User Interface

The GNSS Tracking app uses built-in UI elements from the Android platform. For example, the progress loading screen on app start-up uses the

`android.widget.ProgressBar` class from the Android API. This class contains methods to customize the progress visual indicators for the app. In addition, the display of the world map with the time-lapse slider feature uses the Google Maps API with the `android.widget.SeekBar` class. Details on implementing the Google Maps API are detailed in section 5.3.2.



Startup loading screen | World map

Settings screen | Data refresh screen

Figure 5.3.1 User Interface.

5.3.1 Loading Screen (`AsyncTask`)

The `AsyncTask` object is used in Android to manage tasks that executes in the background and, at the same time, interacts with the graphical user interface (GUI) of

the app [Android, 2015a]. This is to ensure that a time-consuming task that executes in the background does not block the main thread. The GNSS Tracking app uses `AsyncThread` to load data from the SQLite database.

Here are the methods of the `AsyncTask` in detail:

`onPreExecute()`: This is the GUI method that is executed just before the start of the background task. The "loading" message is initialized here.

`doInBackground()`: This is the main method that will be performing the background task of loading data from the SQLite database.

`publishProgress()`: This is the background method that is called from `doInBackground()`. This is used by the app to notify the process of the progress of the background task during the task execution

`onProgressUpdate()`: This is the GUI method is that invoked by `publishProgress()`. This is used by the app to display a "loading" message on the GUI while it is loading the satellite location data from the SQLite database.

`onPostExecute()` and `onCancelled()`: These are the GUI methods that are executed at the end of the task (or when it is cancelled).

Figure 5.3.1.1 illustrates the GNSS Tracking implementation of the `AsyncTask`.

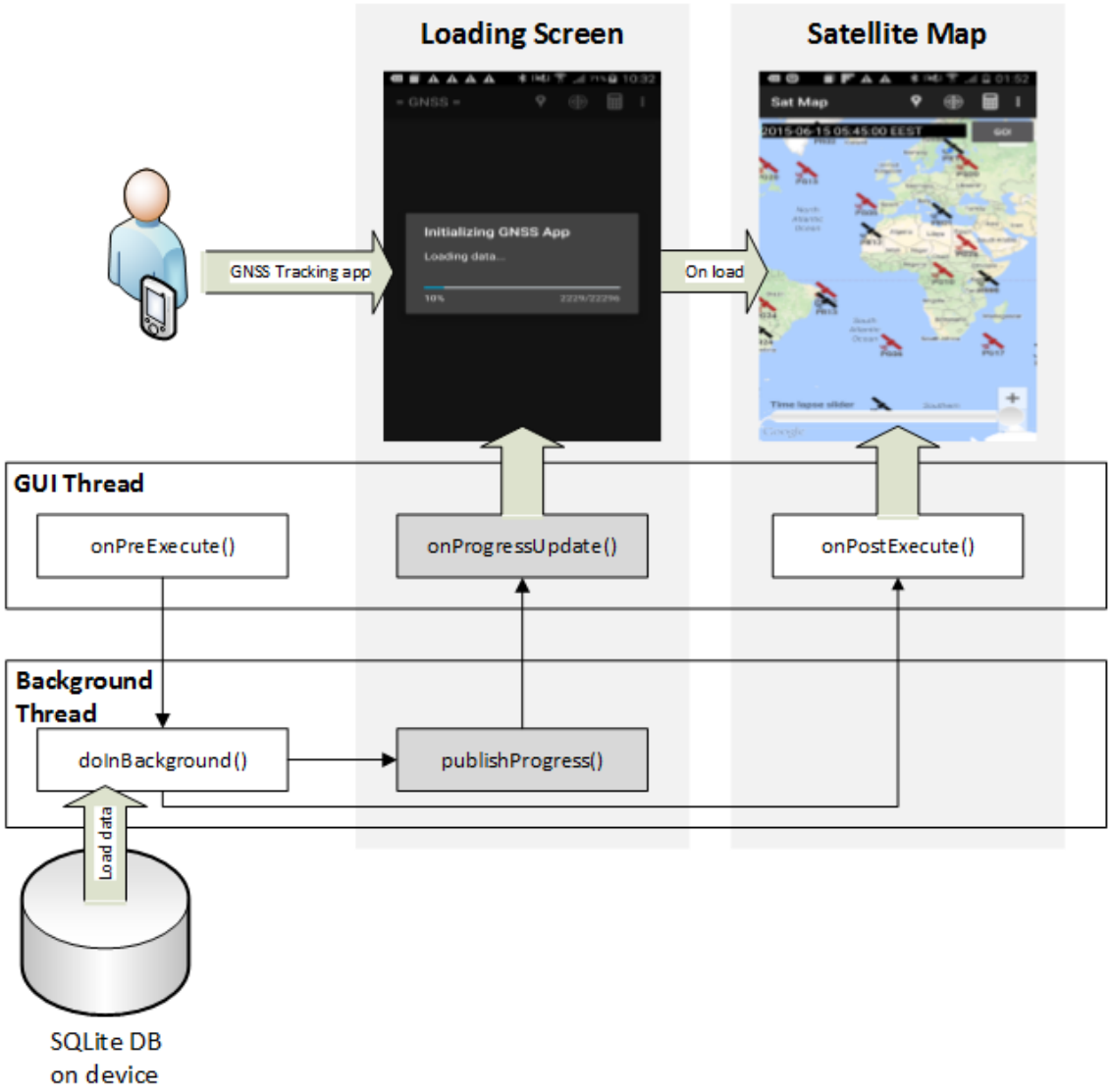


Figure 5.3.1.1 GNSS Tracking app utilization of `AsyncTask`.

5.3.2 Google Maps API

Google provides a free map API for use in various platforms, including web applications, iOS, and Android. Details on this are available on the Google Maps Developer API website at `https://developers.google.com/maps/`. However, the process of using the Google Maps API can be complicated for developers who are expecting to simply include a library in their project for immediate use. First of all, developers will have to register their apps with the Google API Console [Google, 2015b] to allow Google to track usages of their maps API. To register their apps with

the Google API Console, developers must first obtain the developer signature (SHA-1 fingerprint) associated with their developer environment. This is contained in a keystore file located in the `.android` subdirectory of their user directory on their computer. Their SHA-1 fingerprint can also be obtained using a keytool utility provided with the Java SDK. After registering an app, the Google API Console will generate an API key. This key goes into the `AndroidManifest.xml` file. Next, Google Play libraries also need to be listed in the `build.gradle` file under the app directory. Lastly, the map is invoked by the app by specifying it in the user interface layout XML as a `fragment`. A simplified detail of this implementation is specified in Appendix 3.

5.4 SQLite Database

The GNSS Tracking app uses the built-in SQLite database of Android to cache satellite location information. SQLite is an open sourced database. The SQLite library is part of the core Android environment [SQLite, 2015]. The app accesses the SQLite database with the classes and methods within the `android.database.sqlite` library. One disadvantage with using the built-in SQLite database of Android is that the application is limited to the version of SQLite that is shipped with the device [SQLite, 2015]. This is a problem if the app requires a different version of the SQLite database.

The GNSS Tracking app saves the data into the SQLite database on the device when it downloads the satellite location information from IGS. This is to optimize data retrieval upon app start-up. This is because accessing the data on the device is much faster than downloading the data off the third party website. In addition, the coordinate conversion for latitude takes place when the data is retrieved, and saved into the database. This is also much faster than running the conversion calculations every time the data is retrieved from the database. Most of the processing will be deferred to the nightly scheduled job so as to optimize the performance of the app when the user is using the app.

5.5 Satellite Data Download from IGS

The historic satellite location data is obtained from the IGS website. In the past, the IGS have updated their website on a daily basis with files containing tracking data for 24-hour periods in 15-minute intervals. In 2000, this has been changed to an hourly update to satisfy the needs of users who require more real-time data [IGS, 2009]. Therefore it is possible to keep the data on the GNSS Tracking app to retrieve fresh satellite data in hourly intervals. However, due to the performance cost of downloading the data, the

GNSS Tracking app reduces its updates to only once a day, at a time specified by the user.

Downloading the satellite data is a long running process. Therefore, it is set to update as a background task via a scheduler. The download task extends the Android Service class (`android.app.Service`) for this purpose. Since a service does not actually run in a new thread or process from the Activity, it is necessary to create a class to ensure that the download task is executed asynchronously in the background. There are two options for asynchronous execution: (a) to use a Thread/Handler design, or (b) to use the built-in `AsyncTask` class provided with the Android SDK. The GNSS Tracking app opts to use the `AsyncTask` class implementation.

The GNSS Tracking app has a scheduled task that runs every day to request data from IGS, and implements the `AlarmManager` class to do so. The `AlarmManager` class, provided by the Android SDK, is an alarm system that allows events (broadcast intent) to schedule and execute tasks that need to occur even if the application isn't running.

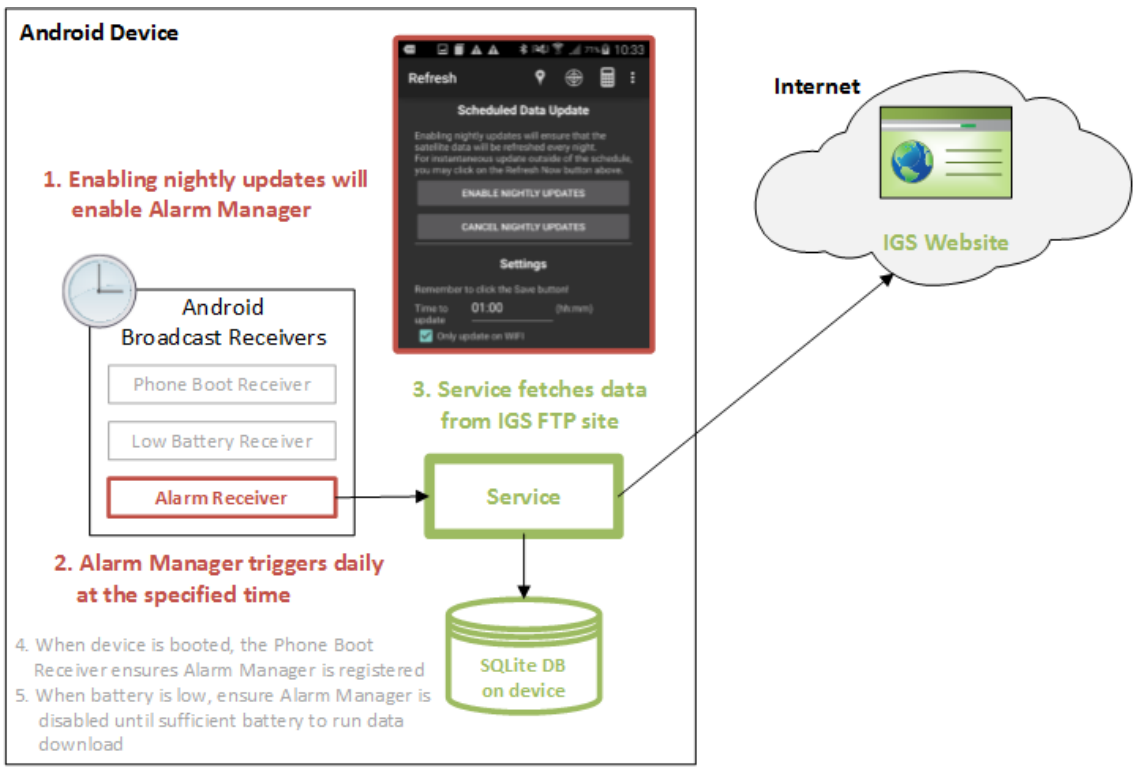


Figure 5.5.1 System diagram which details the historic satellite location data request from IGS.

6 Evaluation

This chapter evaluates the approach taken for this thesis. It begins with a description of the setup of the experimentation, and continues with a presentation of the results of the experimentations. The chapter ends with an evaluation of the experimentations, along with details of its successes and failures.

6.1 Benchmark Setup

Benchmarking is essential for software projects, which include mobile app projects. It is no longer sufficient to test functionality alone. Other areas of an application are equally important, such as app performance, usability, reliability, and ease of maintenance. The objective of benchmarking is to identify and fix problems with the application. This section will describe the benchmark setup for testing the GNSS Tracking app.

As with any system or device testing, the standard procedure for performance testing would be to eliminate inconsistent or misleading results. Before running the tests, there are several precautions that can be taken to ensure that the test conditions were fair and consistent for all experimentations. This will in turn ensure that the benchmark will return meaningful and useful results. First of all, before running any tests, the SIM card was removed from the device. GSM, Bluetooth and automatic updates were disabled. Background apps will be shut down, including Task Manager and alarms, as apps or utilities running in the background may consume system resources and interfere with the benchmarks. Internet connectivity was obtained via Wi-Fi. In addition, factory reset was performed for the device before each set of tests to ensure that the environment was clean and conditions were equal.

As for the test data, all tests were conducted on the same set of data to ensure consistency. Data loads of various sizes were also tested to benchmark the scalability of the app.

During the tests, `System.gc()` was invoked within the code, so that the likelihood of a garbage collector pause during the test execution was reduced. On top of that, the tests were executed repeatedly for enough iteration for the results to be meaningful. The tests were then repeated with a full set of apps installed and running. This is to ensure that the app would also work on a regular device of the average user.

The tests were executed on an official Android version (v4.2.2 “Jelly Bean”) that was shipped with the device, then repeated after upgrading Android to the latest version (Android OS 5.0.1 “Lollipop”). Here are the details of the Android device used to conduct the experimentation portion of the thesis.

Android versions tested:	Android OS, v4.2.2 (Jelly Bean)
	Android OS 5.0.1 (Lollipop)
CPU:	Quad-core 1.9 GHz Krait 300
GPU:	Adreno 320 MHz
Memory:	2 GB RAM
Display:	4.0” 800x480 pixels (233 ppi)
Model number:	Samsung Galaxy S4 GT-I9295
Baseband version:	I9295XXUDOB4
Kernel version:	3.4.0-4101211

Figure 6.1.1 Android device specifications for testing.

6.2 Benchmark Tests

This section describes the areas of the app that will be tested, and the benchmark methods that will be performed.

Algorithm Accuracy: Determine the accuracy of the algorithms used in the conversion for the satellite position data from the Cartesian coordinate system to the geodetic coordinate system.

Benchmark method: The accuracy tests uses the *Gold Data v6.3 Testing For Datum Transformations and Coordinate Conversion Software* from the National Geospatial-Intelligence Agency [NGA, 2014] to validate the accuracy of the algorithm conversion results. The iterations were performed until a change in ϕ is insignificant ($\Delta < 0.000000000001$ decimal degrees). However, it is prudent to keep in mind that the NGA is about geodesy, i.e. extremely precise measuring on the Earth surface, and thus their coordinate conversion validation is more focused on millimetre-accurate conversions near the Earth surface [NGA, 2014]. The same results don’t necessarily hold at the orbital altitude of GNSS satellites, which is at about 20,000 km above the surface of the earth.

Algorithm Performance: Determine the most efficient algorithm for converting the satellite position data from Cartesian coordinates to geodetic coordinates.

Benchmark method: Measure response time and memory usage for execution of different algorithms for the data in bulk. This will be done by running different areas of the app repeatedly against its expected workload and data volume. Experiment with increasingly larger data volumes by increasing the number of GPS weeks, one week at a time.

Availability: Determine the frequency in which the app fails, and anticipate problems.

Benchmark method: Monitor app behaviour and IGS data extraction on a daily basis.

Capacity: Determine if the volume of data required by the app is within reasonable limits for a mobile application.

Benchmark method: Measure storage space used for the data in bulk. Experiment with increasingly larger data volumes by increasing the number of GPS weeks, one week at a time.

Data Download Performance: Determine if the time taken to extract data from IGS is within satisfactory limits.

Benchmark method: Measure response time for the data download. Experiment with increasingly larger data volumes by increasing the number of GPS weeks, one week at a time. Calculate the average time for each data load to retrieve the data.

Database Data Retrieval Performance: Determine if the time taken to extract data from the SQLite database is within satisfactory limits.

Benchmark method: Measure response time for retrieving the data in bulk. Experiment with increasingly larger data volumes by increasing the number of GPS weeks, one week at a time. Calculate the average time for each data load to retrieve the data.

Minimum hardware compatibility: Determine if the app executes well on the minimum hardware defined by the client.

Benchmark method: Test the app on Android OS v4.2.2 (Jelly Bean). Upgrade the device to Android OS v5.0.1 (Lollipop), and test again.

Recoverability: Determine if the GNSS Tracking app stays up and running when errors occur.

Benchmark method: Simulate errors in several areas of the app to determine the measures that need to be taken to have the app recover gracefully.

Reliability: Determine what areas of the app will fail.

Benchmark method: Simulate errors in various areas of the app. For example, simulate loss of internet connection in response from IGS website to determine the behaviour of the app. Other areas to test include the app start-up (with and without the data cache), test all interaction with the SQLite database (insert, select, and delete), test all calculations done within the app, and test all buttons and setting and preference changes in the app.

Usability: Determine if the GNSS Tracking app meets the objectives of the client.

Benchmark method: At the design phase: count number of features to be included in the product. Then at the deployment phase, count number of features delivered. With the discrepancy of requirements, determine if usability of the app is within reasonable limits.

6.3 Benchmark Results and Analysis

6.3.1 Algorithm Accuracy Benchmark Results

The accuracy benchmark results are summarized in Figure 6.3.1.1. The test uses the *Gold Data v6.3 Testing For Datum Transformations and Coordinate Conversion Software* from the National Geospatial-Intelligence Agency [NGA, 2014] to validate the accuracy of the algorithm conversion results. The iterations were performed until a change in ϕ is insignificant ($\Delta < 0.000000000001$ decimal degrees). The relative errors are determined by the results' discrepancy from data from the NGA. Kaplan and Hirvonen and Moritz's methods performed similarly, with the accuracy of Kaplan's at an average relative error of 251.479 mm, and Hirvonen and Moritz's method at an average relative error of 246.396 mm. Torge and Bowring's methods performed much better, with a higher accuracy at an average relative error of 0.00734 mm. Bowring's method only took one iteration to achieve its highly accurate results, while Torge's

method took an average of 3.56 iterations and Hirvonen and Moritz's method took the longest, at an average of 5.42 iterations.

Conversion Method	Iterations	Relative Error (dec. deg)	Relative Error (Distance[1])
Kaplan (closed solution)	N/A	2.51479×10^{-6}	251.479 mm
Hirvonen and Moritz	5.42	2.46396×10^{-6}	246.396 mm
Torge / Heiskanen and Moritz	3.56	7.34465×10^{-11}	0.00734 mm
Bowring	1	7.34031×10^{-11}	0.00734 mm

Figure 6.3.1.1 Average of the relative errors for the algorithm accuracy benchmarking.

It is important to keep in mind that plotting satellite position onto a map on a mobile device is, in itself, a rather inaccurate process. These accuracies may fall under two categories of errors: (a) systematic errors, and (b) computational discrepancies. For systematic errors, we should consider that Google Maps/Google Earth uses WGS-84 datum. Different areas of the world have been mapped or charted on multiple reference systems depending on local preferences at different times. Therefore, even though the WGS-84 datum represents the best modelling of the Earth from a geometric, geodetic and gravitational standpoint, it might not be accurate for some locations on Earth [Dana, 2015]. Another inaccuracy that could possibly originate from Google Maps/Google Earth is that the calibration of Google Maps/Google Earth is not always correct, and this can be visible on the maps where the view is set to the hybrid view to show both the road map and aerial view. In many places the roads from the road map do not lay exactly on top of the roads in the aerial view. On top of that, the source data from IGS could contain inaccuracies as well [Griffiths and Ray, 2008]. This could be caused by discrepancies in the IGS satellite clocks. The accuracy of IGS data is constantly being analysed by the analysis centre coordinator, published at `http://acc.igs.org/`, reported by mail (IGSMAIL-6053), and reviewed at IGS Workshops [Newcastle, 2010].

In terms of computational discrepancies, the coordinate conversion computation routine itself introduced inaccuracies into the data. The `float` and `double` primitive types in Java are commonly used to represent numerical values in applications to conduct arithmetic operations, both of which are floating point representations of

[1] 1 decimal degrees = 100 km = 100,000,000 mm

numbers. However, it is commonly known that rounding errors are inherent in floating-point computation. This is demonstrated in Figure 6.3.1.2, where the accuracy of the results suffers when the value of the "true" latitude demands a higher precision. In the table, the first four rows define a "true" latitude value of -0.000001, to six decimal places. The relative error in this case, was six decimal places. However, it is difficult to form a definite conclusion based on this experimentation alone, as the data from the NGA limits the latitudes with high precision to locations on the equator. One might conclude that the calculations are less accurate near the equator, but there is at least one latitude of -1 which attained a result with a relative error at 12 decimal places. There should be further experimentations done on latitudes with high precision on various locations on the Earth, including the poles and a good distribution of the areas between.

X-Coordinate	Y-Coordinate	Z-Coordinate	"True" Latitude (dec. deg.)	Calculated Latitude (Bowring's Method)	Relative Error (dec. deg.)
6478137.000000	0.000000	-0.112320	-0.000001	$-1.00000351610907 \times 10^{-6}$	$3.51610906994029 \times 10^{-6}$
6678137.000000	0.000000	-0.115810	-0.000001	$-9.9999772407041 \times 10^{-7}$	$2.27592958996035 \times 10^{-6}$
6878137.000000	0.000000	-0.119301	-0.000001	$-1.00000065313833 \times 10^{-6}$	$6.53138329942941 \times 10^{-7}$
6278137.000000	0.000000	-0.108829	-0.000001	$-1.00000049097205 \times 10^{-6}$	$4.90972050011477 \times 10^{-7}$
6678136.999000	0.000000	-115.810264	-0.001000	$-1.0000000037093 \times 10^{-3}$	$3.70930001124259 \times 10^{-9}$
3909067.758000	3909067.758000	3170373.735000	30.000000	29.9999999959876	$1.33746643390017 \times 10^{-10}$
14329035.270000	0.000000	7921997.403000	29.000000	29.000000004365	$1.50517207892900 \times 10^{-10}$
4559797.102000	0.000000	4373636.557000	44.000000	43.9999999956078	$9.98226955640679 \times 10^{-11}$
6224781.003000	0.000000	5981492.822000	44.000000	44.0000000042749	$9.71568684090843 \times 10^{-11}$
8671822.541000	0.000000	-150622.047600	-1.000000	-1.00000000000108	$1.08002495835535 \times 10^{-12}$
110840.007400	0.000000	6307185.029000	89.000000	89.0000000000946	$1.06293999819213 \times 10^{-12}$
111178.584100	0.000000	6326582.074000	89.000000	88.9999999999134	$9.73044366679113 \times 10^{-13}$
773443.522500	0.000000	7316218.504000	84.000000	83.9999999999729	$3.22620237403455 \times 10^{-13}$
198950.226500	0.000000	11355016.100000	89.000000	89.0000000000073	$8.20716777934138 \times 10^{-14}$
120.420626	0.000000	6856752.313000	89.999000	89.9989999999984	$1.76848156990925 \times 10^{-14}$
0.116930	0.000000	6656752.314000	89.999999	89.9999989999997	$3.31586613705676 \times 10^{-15}$
151392.419000	0.000000	8630431.133000	89.000000	89.0000000000002	$2.23541534845874 \times 10^{-15}$

Figure 6.3.1.2 Specific examples of the relative errors for discrepancies in computational data types, using Bowring's method for the calculation of latitude.

Figure 6.3.1.3 examines the accuracy of the conversion algorithms by latitude. The data which suffers from precision loss were omitted from the scatter plots.

Kaplan's method increases in accuracy nearer the equator and the poles. Hirvonen and Moritz's method increases in accuracy as the latitudes approaches the poles. The accuracy for the methods for Torge and Bowring does not seem to be dependent on any coordinate values for accuracy in latitude calculation. The data provided by NGA consisted mostly of positive latitudes, which resulted in the scatter plots being skewed on the positive latitude. Therefore, the scatter plot analyses are deemed inconclusive.

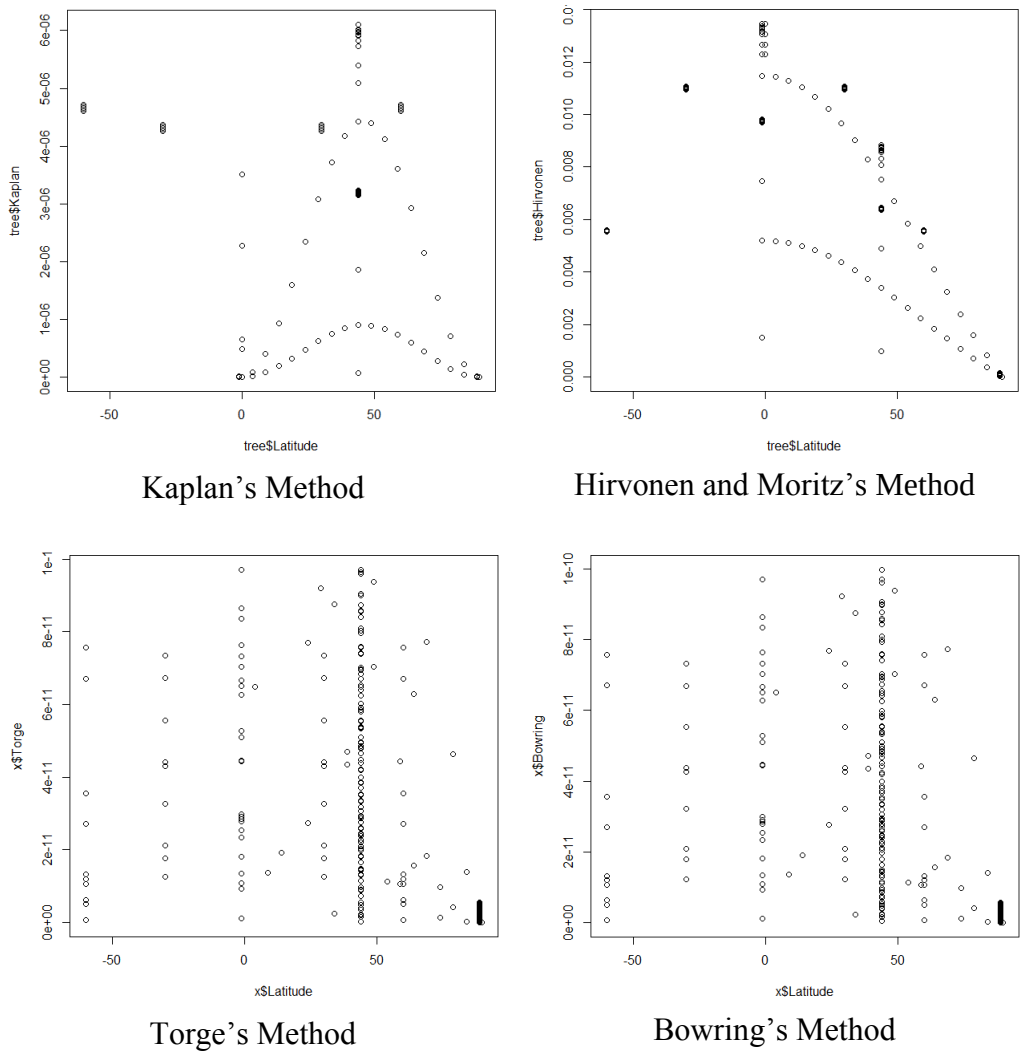


Figure 6.3.1.3 Scatter plot to examines the accuracy of the conversion algorithms by latitude. The x-axis is the latitudes, and the y-axis is the relative errors.[2]

[2] Note that the results for Kaplan and Hirvonen and Moritz mostly congregate on the right side of the scatter graph because the test data provided by the NGA were mostly ranged in the positive latitudes.

As a solution to the rounding errors in floating point computation, Java has a `BigDecimal` class which will handle very large numbers and very small numbers. To determine the magnitude of the discrepancy between the `double` and the `BigDecimal` data type, calculations on both data types were conducted and the results were compared. For this test, Kaplan's method was used because it contains only one trigonometric function, which reduced the need for third party libraries for conversion between the `double` and `BigDecimal` data types, which might introduce precision loss into the experimentation. The calculations were repeated using R, a language for statistical computing [R, 2015]. The inclusion of a calculation using R is to exhibit that Kaplan's method is capable of results with high precision when rounding errors were not involved. The results of the tests are summarized in Figure 6.3.1.4. It shows that the `double` data type caused a loss in precision with a relative error of 475 mm. By using the `BigDecimal` class, some of the precision loss was salvaged and it returned a smaller relative error of 0.00058 mm. However, the `BigDecimal` class suffers in computational costs, as shown in Figure 6.3.1.5. The increase in time for the `BigDecimal` data type is caused mainly by the constant garbage collection being performed to recover the high memory consumption of using the data type. Bowring's method, calculated using the `double` data type, returns a sufficiently accurate result with a relative error of 0.00132 mm.

Data Type Test	Computed Latitude	Relative Error (dec. deg)	Relative Error (Distance)
Kaplan's method with `double` data type	-59.99971498362769	4.75027×10^{-6}	475 mm
Kaplan's method with `BigDecimal` class	-60.00000000035068	5.84467×10^{-12}	0.00058 mm
Kaplan's method with R	-60.00000000000677	1.12833×10^{-13}	0.0000112 mm
Bowring's method with `double` data type	-60.00000000079428	1.3238×10^{-11}	0.00132 mm

Figure 6.3.1.4 Relative error for discrepancies in computational data types. "True" value of computed latitude = -60. Latitudes were calculated from Cartesian coordinates of $x = -2246552.197953,\ y = -2246552.197953,\ z = -5465836.117787.$

When taking into consideration the computational cost in terms of time and memory spent, Bowring's method with the `double` data type far outperforms Kaplan's

`BigDecimal` results. This can be seen in Figure 6.3.1.5 below. The Java code implementation for the data type discrepancy tests can be found in Appendix 4.

Data Type Test	Relative Error (dec deg)	Relative Error (Distance)	Time (ms)	Memory (MB)
Kaplan's method with `BigDecimal` class	5.84467×10^{-12}	0.00058 mm	535537	38.774
Bowring's method with `double` data type	1.3238×10^{-11}	0.00132 mm	0.793457	26.264

Figure 6.3.1.5 Comparison of computational duration and memory costs between Kaplan's method with `BigDecimal` class and Bowring's method with `double` data type.

6.3.2 Algorithm Performance Benchmark Results

The efficiency of an algorithm depends on two factors, (a) the number square roots, cube roots, and trigonometric functions requiring evaluation and (b) the number of iterations required for the indirect methods [Gerdan and Deakin, 1999]. Figure 6.3.2.1 tabulates the number of squares, roots, and functions in the four coordinate conversion algorithms. From the table, Hirvonen and Moritz's method is the least efficient in terms of the number of functions to be executed, due to having to iterate on the average of 5.42 times before converging to a solution. Bowring's method is the most efficient method, with only one iteration of 27 functions.

Conversion Method	Average num of iterations	Num of x^n	Num of $\sqrt[n]{x}$	Num of trigonometric functions	Total
Kaplan (closed)	1 (closed)	34	7	1	42
Hirvonen and Moritz	5.42	7	2	6	81.3
Torge / Heiskanen and Moritz	3.56	6	2	5	46.28
Bowring	1	18	3	6	27

Figure 6.3.2.1 Algorithm efficiency matrix. For methods with multiple iterations, the total multiples the number of functions with the number of iterations

The performance benchmark was executed on 600 records, over 10 iterations. As per the efficiency formula, Figure 6.3.2.2 confirms that Bowring's algorithm is the fastest and most accurate method. The method by Hirvonen and Moritz takes the longest to execute at three iterations. Bowring's method attains accuracy at a single iteration, therefore it is not an issue that a comparison of the methods by a single iteration shows that Bowring's method is costly due to the multitude of trigonometric functions used.

Conversion method	Average no. of iterations	Relative Error (dec. deg)	Average time (ms)	Memory spent (MB)
Kaplan (closed)	1 (closed)	2.51479×10^{-6}	23.3947754	19.908
Hirvonen and Moritz	5.42	2.46396×10^{-6}	4.0039062	19.579
Torge / Heiskanen and Moritz	3.56	7.34465×10^{-11}	2.6306152	19.583
Bowring	1	7.34031×10^{-11}	2.0202636	19.593

Figure 6.3.2.2 Algorithm performance benchmarking on a set of 600 records.

Summarizing our algorithm performance benchmark, Bowring's method is the most suitable for the GNSS Tracking app. As stated before and verified by the tests conducted, only one iteration of Bowring's method is required for a sufficiently accurate result [Bowring, 1985]. Since the difference in memory spent between the methods is negligible, Bowring's method is selected for its accuracy and performance.

6.3.3 (Other Areas) Benchmark Results

Availability: The app was monitored for the frequency in which the app fails, and for problems in operation. The daily IGS data download was also monitored. There were some availability issues observed with the app usage, and these are documented in Figure 6.3.3.1 below. During the testing and debugging phase, the availability issues were fixed. However, the data download process with the IGS website was still constantly causing issues. The number of issues that had surfaced over the course of the development work is a forecast of further issues to come with the service. For example, IGS migrated their data from `HTTP` to `HTTPS`, and then again to `FTP`, all within a six month period. Switching from `HTTP` to `HTTPS` resulted in a `javax.net.ssl.SSLHandshakeException` when the IGS

directory for the data download was encrypted with SSL. Due to time constraints, migration from the `HTTPS` to the `FTP` URL was not implemented. This is documented in the Future Work section of this thesis.

Problem	Status
IGS data download took too long to complete	**Resolved.** Initially, the data download took over four hours to complete for two days' worth of data. A bulk-insert SQLite method was implemented and the download metric was switched from days to GPS weeks. This is because the files on the IGS website are organized in folders by GPS weeks. Now the code can grab all the files from a single folder and avoid checking the dates after each file download. The performance was vastly improved to less than one minutes for the same amount of data download.
IGS data download would stop running	**Resolved.** This happens when the download takes too long to complete. The process would time out or the device would run out of memory. The solution was to save only the minimal amount of data needed. Parameters including the xyz-coordinates, GPS week numbers, the .sp3 file name, and other data from IGS were dropped. Only the latitudes and longitudes were needed for the app to work properly. Switching to the bulk-insert SQLite method helped as well. Other code optimizations were implemented.
App would crash	**Resolved.** App was throwing runtime exceptions. This was caused by errors in variable assignments in the UI layout files. These were fixed.
NaN errors	**Resolved.** There were some NaN (Not a Number) errors with the Bowring algorithm conversion at latitude decimal degree -90, 0, and 90, which are the latitudes for locations at the poles of the Earth. NaN values were eliminated from the results, as the satellites do not orbit at the poles. Other errors in the calculations and input errors from the .sp3 files were also trapped and handled.
Out of Memory errors with loading app	**Resolved.** The app will run out of memory if too much data has been downloaded and attempted to be loaded. There is currently an imposed limit to the amount of data a user can download, which is a maximum of three GPS weeks of data. The data structures were also trimmed to only store the minimal needed parameters.

Figure 6.3.3.1 App availability issues and status.

Capacity: The volume of data required by the app is within reasonable limits for a mobile application, as seen in Figure 6.3.3.2 below. The experimentation involves

downloading and using some apps for games and networking, and comparing the data storage usage between these apps and the GNSS Tracking app.

Num of GPS Wks	Num of Files	Num of Records	Storage (MB)
1 week (7 day)	52	209,088	23.91
2 weeks (14 days)	108	429,792	33.68
3 weeks (21 days)	164	657,312	43.39

Figure 6.3.3.2 Total storage capacity utilized by the GNSS Tracking app. Capacity is displayed in megabytes (MB). This sample includes data for both GPS and GLONASS systems.

With two weeks' worth of data, the GNSS Tracking app occupies under 50 MB of storage capacity, which is well below that used by some popular apps, as seen in Figure 6.3.3.3 below.

Popular mobile app	Size
GNSS Tracking app (3 weeks of data)	43.39 MB
Amazon Kindle	154 MB
Clash of Clans	59.93 MB
Crossy Road	129 MB
Skype	101 MB

Figure 6.3.3.3 Comparison with popular apps.

Data Download Performance: This test was conducted over Wi-Fi with an internet connection download speed of 1.84 Mbps. In this test, the experimentation determines if the time taken to extract data from IGS is within satisfactory limits. It takes multiple files from IGS to fill the historic data cache for the satellite data. It takes about five to 10 seconds to process one file from IGS, including downloading the file, uncompressing and reading it, and saving the data into the SQLite database. One weeks' worth of data cache for two satellite system (GPS and GLONASS) takes about 52 files, which takes about 5 minutes and 44 seconds

to complete the data download, at optimal conditions[3]. This is well within the acceptable time performance for a data download process that executes in the background of a mobile device, even if the data download is expected to occur on a daily basis.

Num of GPS Wks	Num of Files	Num of Records	IGS Download Time	App Load (secs)
1 week (7 days)	52	209,088	5 mins 44 secs	0.505
2 weeks (14 days)	108	429,792	11 mins 15 secs	0.429
3 weeks (21 days)	164	657,312	17 mins 44 secs	0.585

Figure 6.3.3.4 Performance of the GNSS Tracking app. Data is averaged over five tests. This sample includes data for both GPS and GLONASS systems.

App Usage Performance: The experimentations measured response time for retrieving the data in bulk. The average time to load the app on start-up, including the time is takes to retrieve the data for one weeks' worth of data, is 0.505 seconds. This is within acceptable limits of a mobile app. In addition, a progress loading screen is implemented to ease the user through the app loading process.

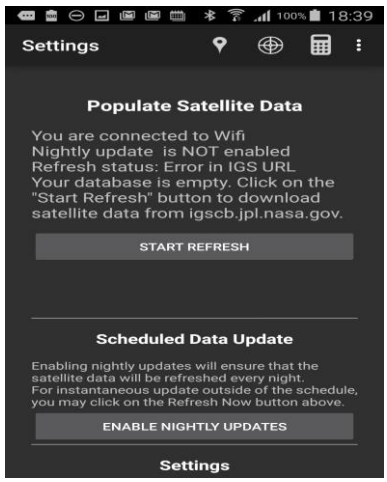
Minimum hardware compatibility: The app was implemented and tested with Android OS 4.2.2, which was the minimum hardware compatibility requested by the client. User experience of the app was sufficiently pleasant during testing. The app loads within a reasonable time limit, and it does not crash or freeze when using it.

Recoverability: Errors in several areas of the app was simulated. In general, the data download was the section that was consistently throwing errors. These errors were either resolved, or avoided by redirecting the app process flow. These are documented in Figure 6.3.3.1 above. The other sections, i.e. world map, sky plot, settings, and refresh screens, operated smoothly.

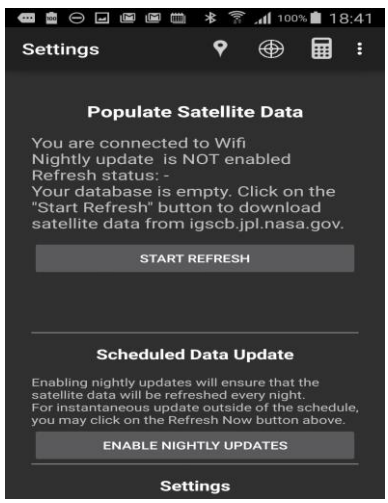
[3] This test was conducted over Wi-Fi with an internet connection download speed of 1.84 Mbps. Optimal conditions involve not running other apps which would consume device resources, and would defer processing of the data download.

Reliability: The experimentations simulated loss of internet connection and errors in response from IGS website to determine the behaviour of the app. These errors, and any other exceptions thrown in the course of using the app, were caught and handled within the app, and the user was notified. Some examples of possible error scenarios are documented below, in Figure 6.3.3.5.

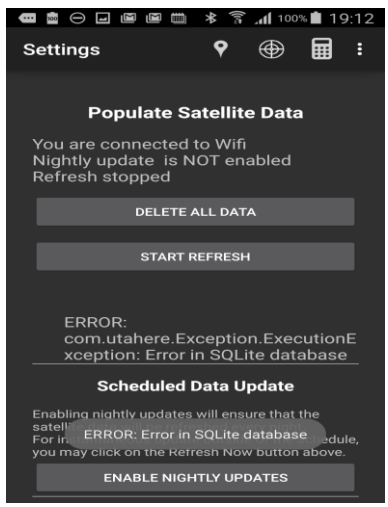
Error when device is not connected to Wi-Fi



Error when URL for IGS is not working



User is automatically redirected to Refresh screen when there is no data



Error when reading the SQLite database

Figure 6.3.3.5 Notifications to the user when errors and exceptions were experienced during the course of using the app.

Usability: Basically, the GNSS Tracking app meets the objectives of the client by delivering all the main requirements. However, some of the details were omitted due to time constraints, and quality was prioritized over quantity in terms of requirements to be implemented. In addition, there were some basic user features implemented that were not defined by the client. These are detailed in Figure 6.3.3.6 below. Usability of the app was overall acceptable. The time performance for the data download from IGS may take longer than that which a user is willing to sit around and wait for completion. This is resolved with having the data download running as a background thread, at a time specified by the user. With the daily scheduler responsible for downloading data from the IGS website, this process is concluded to be acceptable for a good user experience.

ID	Requirement	Delivered
Preferences		**Yes**
F-1	User can choose which satellite systems are visualized on the UI	Yes
F-2	The user can set an arbitrary location as the reference location by inserting the WGS-84 coordinates (latitude and longitude in degrees/minutes/seconds)	No
F-3	The user can set an arbitrary time as the reference time by inserting the time in UTC+00 (year/month/day/hour/minute/second) and the UTC TimeZone. Future predictions limited to +2 days from the current time.	No
	Total requirements delivered	**1 of 3**
Background requirements		**Yes**
F-4	The application uses the current location and time (UTC+TimeZone) of the device as the default location and reference time	Yes
F-5	The application will visualize on the UI the UTC-time and the UTC TimeZone	Yes
F-6	The application will update the constellation statuses (Skyplot, Worldmap) at the default update rate (i.e. UI refreshed e.g. at 1Hz)	Yes
F-7	The application will automatically download a new set of satellite ephemerides at the most convenient update rate to keep the satellite location and health information valid. Health information can be read/parsed from NANU/NAGU reports	Yes
	Total requirements delivered	**4 of 4**
World map		**Yes**
F-8	User can freeze the UI by pausing the updates	No

F-9	The application will indicate the overall health of the satellites e.g. by color coding	No
F-10	The application will indicate the reason of the satellite health/unhealth status by a color/number/text or by allowing the user to reveal more status information by pointing/touching the satellite icon on the UI	No
F-11	User can zoom in/out the world map	Yes
	Total requirements delivered	**1 of 4**
Sky Plot		**Yes**
F-12	User can modify the reference time e.g. by a slider to see the changes in the Skyplot (fast-forward/fast-backward)	No
F-13	User can modify the reference time e.g. by a slider to see the changes in the Skyplot (fast-forward/fast-backward) in the "pause-mode". Future predictions limited to +2 days from the current time.	No
F-14	User can select and set the reference location for the Skyplot by touching the world map	No
	Total requirements delivered	**0/3**

Figure 6.3.3.6 Breakdown of client requirements implemented and delivered.

Nine additional features were included, as detailed in Figure 6.3.3.7 below. These features improve the usability of the app by allowing the user to control more of the settings of the app via the UI. A common alternative would be to change the settings in the code and redeploy the compiled APK, which would be a less efficient process.

Additional Features Implemented	
Loading screen	Yes
Calculators	Yes
Option to update URL for IGS data source via the app UI	Yes
Option to update number of days to cache data via the app UI	Yes
Option to turn on/off daily data sync via the app UI	Yes
Option to change time for daily data sync via the app UI	Yes
Option to only download data on Wi-Fi connection via the app UI	Yes
Total requirements delivered	**7 of 7**

Figure 6.3.3.7 Breakdown of additional requirements implemented and delivered.

6.4 Analysis of Implementation

Memory consumption and execution times were around the same range for all four algorithms. However, Torge's and Bowring's method outperformed Kaplan's and Hirvonen and Moritz's methods in terms of accuracy with an average relative error of 0.00734031 millimetres, and Bowring's method alone outperformed all the other three methods with an average duration of 2.0202636 milliseconds for 600 records. There were inaccuracies in the code introduced by floating point rounding errors, but Bowring's method is sufficiently accurate and efficient for the purpose of the GNSS Tracking app.

In terms of the performance analysis, the management of large data sets and the performance of sorting algorithms have always gone hand-in-hand. However, this was not the case for the GNSS Tracking app. Data from the IGS website was retrieved and directly inserted into the built-in SQLite database of the device. For display on the world map of the app, data from the database was retrieved and stored into a `HashMap` with the date and time as keys for the map. There was no need for the data to be sorted in any order for it to be useful. As a result, performance relating to sorting was not an issue with the GNSS Tracking app.

Another area of concern with regards to the performance analysis was the data download from IGS. The methods used to download the data from IGS could be further researched and improved upon. The implementation for the GNSS Tracking app for this thesis resulted in a time-consuming data download process that is deferred to a nightly-executed scheduled job running in a background thread. While a 5 to 20 minute (depending on the number of days' cache requested by the user) nightly background job might seem time consuming to some mobile users, there are options to optimize this experience. For example, the user may select to cache less number of days of data, or to have the data download scheduled to run nightly. The user also has the option to shut off the automatic data download completely, and utilize the manual refresh feature. There is still room for improvement, and this can be further optimized in a future release of the app.

6.5 Quality of Benchmark and Evaluation of Results

In terms of coverage, the benchmark tests covered both functional and non-functional aspects of the GNSS Tracking app. This is sufficient for the purpose of this thesis. The tests also covered the performance of the app extensively, through different amounts of data and usage, as well as repeated iterations of the same tests over several days, at different times of the day. On top of that, the tests were performed on a "clean" device

with a fresh factory reset, as well as on a device fully loaded with various apps running in the background. These tests provided a suitably wide range of scenarios for which the app would be utilized in a real world situation.

The results were quite expected and satisfactory. All of the tests yielded consistent, predictable results. The performance of the app was within acceptable levels. This concludes that the app is consistently performing in a reliable, predictable way.

However, the benchmark may be lacking due to the fact that the tests were conducted only on one device, which is a Samsung Galaxy S4. Due to limited resources, there were no additional tests performed on other devices. In addition, there were no tests performed on effect of the power usage of the app. There were also no investigation conducted on when and where the app allocates and releases memory, or where the memory consumption of the app is the highest.

6.6 Failures and Limitations

While all research suffers from limitations, it is important to reflect and identify them in the scope of the research. It is acknowledged that while the thesis had accomplished its research goals and the GNSS Tracking app has been implemented and delivered to the client, there were still a few failures identified. This section aims to explain the nature of these failures and limitations, and suggest the methods in which they can be overcome in the future. They are also given a mention in the Future Work section at the end of this thesis for completeness.

Limited testing performed: Although extensive and intensive testing for performance and accuracy was conducted during the development phases on the app itself, there was no testing done by a variety of end users. Due to time constraints, testing was limited to the developer's point of view, and a demo was given to the thesis supervisor. This thesis acknowledges that further issues could be identified and resolved if more testing had been conducted by a variety of users and devices, especially users who would have directly benefited from the development of the GNSS Tracking app. There were also no additional tests performed on other devices aside from one Samsung Galaxy S4. In addition, there were no tests performed on effect of the power usage of the app. There was also no investigation conducted on when and where the app allocates and releases memory, or where the memory consumption of the app is the highest. Further testing should be conducted to improve the app.

Limited research due to broad research scope: The thesis topic consisted of two parts – a research on the algorithm analysis for coordinate conversion, and a research on the performance of an Android app. Due to the broad scope of the two-pronged research topic, these two areas could have been delved in more deeply than what had been accomplished. For example, further understanding of the research algorithms could have produced a new solution, or a deeper understanding of the performance issues of Android and Java could have optimized the app further.

Undelivered client requirements: The app failed to meet all of the requirements of the client. This included a time-lapse slider for the sky plot feature. Even though a time-lapse slider was implemented for the world map feature, having a time-lapse slider for the sky plot would have helped user identify satellites in view over a given location over a period of time. In addition, the client requested to have the .sp3 files from IGS parsed for satellite location health information and have this information reflected on the app. This was also not delivered due to time constraints. A detailed breakdown of these features is documented in the Usability review above, in Figure 6.3.3.5.

GNSS Tracking app limitations: Testing of the app has revealed some shortcomings. First of all, the app is incapable of functioning with more than three weeks, or 21 days of satellite data cache. With more than three weeks' worth of data, the app crashes upon start-up due to the lack of memory. In addition, the time lapse slider operates with increasing delay as the amount of data increases. Lastly, the IGS site used for the data downloads needs to be updated to the IGS-recommended `FTP` site. It is currently using the out-dated `HTTPS` site.

7 Conclusions and Future Work

This chapter covers the conclusion of the research for this thesis. First, it answers the research questions presented at the beginning of this thesis. Next, it discusses the conclusions derived from the literature review and experimentations performed for this thesis. It will also summarize the contributions to the field by this thesis. Lastly, it will present some possible future work that can extend the research for this thesis.

7.1 Answers to Research Questions

As verified by the experimentations, it is concluded that it is feasible to implement a GNSS Tracking app on the Android platform. The research questions presented at the beginning of this thesis will be answered below.

How does the GNSS tracking app help Here.com?

Here.com had indicated a need for an open-sourced application that displays a multi-day cache of historic satellite location data on a map. The map should be able to track the path of the satellites over a period of several days on a time lapse slider. As a result of the work for this thesis, the app has been implemented and delivered to the client. Even though not all of the requirements have been delivered, but the key features of the app have been implemented. Here.com is able to build upon the work for this thesis and enhance the app as needed.

How does the GNSS tracking app contribute to the field of GPS technology?

The research contributes a new tool to the field, which is an open-sourced GNSS Tracking app for the Android platform. The code is shared on GitHub at `https://github.com/UTA-Here/GNSSTAM` for future developers to further customize and improve the app. The app may not change the way people work, but will broaden the available options for those who wishes to have such an application for Android mobile devices. It provides a method for users to track the location of satellites over the period of several days via a time-lapse slider

How is the GNSS tracking app affected in terms of resource usage, performance, and usability?

The resource usage in terms of storage space for a cache of two weeks of satellite data is well below the average for a mobile app. Performance of the app is acceptable for a three-week data cache for the data download from IGS, but anything more than three weeks will result in poor performance for the app. The

usability of the app itself is pleasant and well-balanced due to the implementation of the client's requirements as well as some additional usability features that were not requested by the client.

What new knowledge was derived from the coordinate conversion algorithm benchmarks?

This thesis contributes to the field a study on the algorithmic accuracy and system performance analysis for GPS coordinate conversion on the Android platform. Developing for the Android platform involves using a high level programming language such as Java or C#, both of which suffers from the loss of precision with floating point data types. There are solutions for higher precision, such as using the `BigDecimal` data type or using another language on the Android platform such as Scala. However, these workarounds suffer in performance, which is not optimal for an app that depends on a large amount of data to operate. For the tests conducted on the four algorithms, Bowring's method was the most optimal solution in terms of accuracy of up to an average relative error of 0.00734031 mm, even when using the Java `double` data type. The conclusion of the high accuracy of Bowring's method is in agreement with previous experimentations and research conducted on coordinate conversion algorithms. The performance of Bowring's method on the Android platform is also the most optimal when compared to the other methods. Therefore, Bowring's method was selected for the purpose of the GNSS Tracking app.

What are the critical limitations of running the GNSS tracking app on a mobile device?

The key part of the app, which is loading the app at start-up and operating the time-lapse slider of the world, operates well within normal response time when the cache of satellite data was limited to three weeks or less. However, downloading the data from IGS increases in time and energy consumption as the data cache increases, and the app crashed on start-up while loading the data when the data cache exceeded 21 days. Therefore, the GNSS Tracking app is limited to a data cache of 21 days or less. The exact amount of data cache required for a GNSS Tracking app to be useful is not known, but it is expressed by the client that "several days" of data is desired. In conclusion, while the app meets the client's requirements of having several days cache of satellite data, it would be beneficial to explore the possible optimal amount of data for the field in a future research work.

7.2 Conclusions

In addition to the algorithm and performance analysis, the goal of this thesis is to evaluate the capabilities of the Android mobile device to support the requirements of a satellite tracking application.

As far as the research for this thesis has shown, there is not an open-sourced Android app on the market that contains a time-lapse slider view of historic satellite locations. Therefore, the work for this thesis contributes an open-sourced GNSS Tracking app for the Android platform. The code is shared on GitHub at `https://github.com/UTA-Here/GNSSTAM` and available for download and customization by anyone interested in the field.

In the literature review that was conducted for this thesis, there was also no published research found on accuracy and performance benchmarks for geographical coordinate conversion algorithms on the Android platform. Therefore, this thesis contributes to the field a study on the algorithmic accuracy and system performance analysis for GPS coordinate conversion on the Android platform.

The benefits of implementing a GNSS Tracking app include an opportunity to study the path of satellite locations via a commonly-used device such as a mobile phone. The result of the experimentations conducted on the GNSS Tracking app prototype led to improvements made for the final product. These improvements included enhancements in usability and performance. In addition, the knowledge gleaned from the research of this thesis would help the construction of better and more accessible tools in the field of location-based services.

For the accuracy benchmarking, the tests conducted on the four algorithms concluded that Bowring's method was the most optimal solution in terms of accuracy of up to an average relative error of 0.00734031 mm, even when using the Java `double` data type. The conclusion of the high accuracy of Bowring's method is in agreement with previous similar experimentations and research conducted on coordinate conversion algorithms. There are solutions for higher precision, such as using the `BigDecimal` data type or using another language on the Android platform such as Scala. However, these workarounds suffer in performance.

The performance benchmarking concluded that the GNSS Tracking app performs optimally for a cache of less than 21 days of satellite data. The three main performance points of the app include the data download from IGS, loading the app on start-up, and operating the time-lapse slider. The time taken to load the app on start-up and to operate the time-lapse slider with 21 days of satellite data was deemed operational at a satisfactory performance speed. The main issue is with the data download process from IGS. It takes approximately five to 10 seconds to process one file from IGS, including downloading the file, uncompressing and reading it, and saving the data into the SQLite

database. For the app to be useful, at least two weeks of satellite data is needed, which takes over five minutes to download and process. This is within an acceptable time limit for retrieving data from a third party website whose data is offered in UNIX compressed .Z files. The data also have to be refreshed and kept updated on a daily basis. This is accomplished by a nightly scheduled job. All in all, for a mobile app to require a daily five to 20 minute data download, while still within an acceptable time limit, might be a usability issue for some users. Further work should be undertaken to determine if this is an acceptable factor in such an app, or if there are additional ways of optimizing the process.

This thesis concludes that it is feasible to implement a GNSS Tracking app on Android. There are many factors in the determination of a good mobile app, such as usability, good functionality, ease of maintenance, and good performance. The app satisfies all of these requirements. For scientific tools, accuracy in the data computation is a valuable factor as well. The data for the GNSS Tracking app attains accuracy via the Bowring coordinate conversion method. The research was successful in that the GNSS Tracking app was completed and delivered. The app is pleasantly usable and accomplishes what it was designed to do. The work for this thesis allows future research to be continued on this tool.

7.3 Future work

There is an idea about good software that it is never completed; only abandoned. Stagnant software is helpless to a myriad of external factors, such as OS upgrades, ever-changing APIs and libraries, or even to changing user interests and needs. While the work for this thesis have been completed, there will always be possible enhancements and improvements to be implemented for the GNSS Tracking app. It is in the best interest of the field to identify and document possible flaws and shortcomings of the research and the app, and where it could be improved upon. This section details the possible future work for this research.

Algorithm Accuracy Enhancements

Research other workarounds to the algorithm precision issue. Implement the coordinate conversion methods in another programming language on the Android platform, such as Scientific Python [SciPy, 2015], the GNU Multiple Precision Library [GNU, 2014], the GNU MFPR Library [GNU, 2015], or Scala [Scala, 2015].

Performance Enhancements

Future enhancements can be undertaken to expedite the data download process from IGS. For example, there could be a separate `cron` job running on an external server to download and uncompress the .Z files from IGS. This may optimize data processing on the mobile device. There is also a possibility to explore the cost of maintaining an external server as a worthy investment to redeem some time loss in interacting with the IGS servers. In addition, further research could be conducted on the amount of data cache is required for those involved in related fields.

Future predictions for satellite locations

One of the requirements of the client is to have the time-lapse slider display satellite locations at a user-specified future date. The data from IGS does contain some future positions, predicted at one or two days ahead of the current date and time. Therefore, part of this future enhancement would also be to mark which future data is from IGS, and which is calculated. In addition, future data in the database will have to be overwritten with historic data from IGS as the data is downloaded.

Below is the equation to interpolate the coordinates of the satellites using a set of existing coordinates.

$$future\_pos = pos1 + ((time - time1)/(time2 - time1))x(pos2 - pos1)$$

Enhancements for Sky plot

Another requirement of the client that was dropped was to have a time-lapse slider for the sky plot. The current sky plot simply displays the satellites overhead for the current location of the user. Future enhancements could be implemented to include a time-lapse slider for historic satellite locations, as well as a manually-entered user location.

Use FTP site instead of HTTP site for IGS data

Currently, the GNSS Tracking app points to the HTTP location for the IGS data. However, it should be pointing to the FTP location instead.

GPS:

HTTP URL: `https://igscb.jpl.nasa.gov/igscb/gps/`

FTP URL: `ftp://ftp.igs.org/pub/gps`

GLONASS:

HTTP URL: `https://igscb.jpl.nasa.gov/igscb/glonass/products/`

FTP URL: `ftp://ftp.igs.org/pub/glonass/products/`

Display more information from IGS

It is possible to determine the health of the satellite location data retrieved from IGS [Griffiths and Ray, 2008]. Currently, touching the satellites on the map would toggle its vehicle id and its latitude and longitude. Adding the health information would be a useful parameter to view as well.

Data for other satellite systems – BEIDOU, GALILEO, and other satellite systems

Currently, IGS only has location data for the GPS and GLONASS systems. Displaying the data for other satellite systems would enhance the usability of the GNSS Tracking app.

More conversion methods – Borkowski, Fukushima, Lin and Wang, Toms

Other algorithms could be implemented and benchmarked. Perhaps the code could store execution times of the algorithms and toggle to a faster conversion method based on known speeds of existing conversion times for the algorithms.

Port the app onto iOS and Windows devices

Another possible future work would be to port the GNSS Tracking app onto other platforms, such as iOS or Windows.

References

[Agrawal and Wasserman, 2010] Agrawal, S. and Wasserman, Anthony I.. *Mobile Application Development: A Developer Survey*. Carnegie Mellon Silicon Valley, 2010.

[Android, 2015a] Android. *Android developer documentation*. [API 22] Available at: `http://developer.android.com/reference/android/os/AsyncTask.html` [Accessed 18 Aug. 2015].

[Android, 2015b] Android. *Android developer documentation*. [API 23] Available at: `http://developer.android.com/reference/android/location/GpsStatus.html` [Accessed 18 Jul. 2015].

[Android, 2015c] Android. *Android Design Principles*. 2015. Available at: `http://developer.android.com/design/get-started/principles.html` [Accessed 18 Jul. 2015].

[Android, 2015d] Android. *Android NDK*. 2015. Available at: `http://developer.android.com/tools/sdk/ndk/index.html` [Accessed 18 Jul. 2015].

[Android, 2015e] Android. *Android Dashboards*. 2015. Available at: `http://developer.android.com/about/dashboards/index.html` [Accessed 22 Jul. 2015].

[Android, 2015f] Android. *Performance Tips*. 2015. Available at: `http://developer.android.com/training/articles/perf-tips.html` [Accessed 4 Sept. 2015].

[Android, 2015g] Android. *Android version 5.0 Behavior Changes*. 2015. Available at: `https://developer.android.com/about/versions/android-5.0-changes.html` [Accessed 27 Jul. 2015].

[Android, 2015h] Android. *Application Fundamentals*. 2015. Available at: `http://developer.android.com/guide/components/fundamentals.html` [Accessed 1 Sept. 2015].

[Android, 2015i] Android. *App Manifest*. 2015. Available at: http://developer.android.com/guide/topics/manifest/manifest-intro.html [Accessed 1 Sept. 2015].

[Android, 2015j] Android. *Keeping Your App Responsive*. 2015. Available at: http://developer.android.com/training/articles/perf-anr.html [Accessed 5 Sept. 2015].

[Android, 2015k] Android. *Manipulating Broadcast Receivers On Demand*. 2015. Available at: http://developer.android.com/training/monitoring-device-state/manifest-receivers.html [Accessed 5 Sept. 2015].

[Android, 2015l] Android. *Determining and Monitoring the Connectivity Status*. 2015. Available at: http://developer.android.com/training/monitoring-device-state/connectivity-monitoring.html [Accessed 5 Sept. 2015].

[Android, 2015m] Android. *Monitoring the Battery Level and Charging State*. 2015. Available at: http://developer.android.com/training/monitoring-device-state/battery-monitoring.html [Accessed 5 Sept. 2015].

[Android, 2015n] Android. *Loading Views On Demand*. 2015. Available at: http://developer.android.com/training/improving-layouts/loading-ondemand.html [Accessed 5 Sept. 2015].

[Android, 2015o] Android. *Making ListView Scrolling Smooth*. 2015. Available at: http://developer.android.com/training/improving-layouts/smooth-scrolling.html [Accessed 5 Sept. 2015].

[Android, 2015p] Android. *Optimizing Layout Hierarchies*. 2015. Available at: http://developer.android.com/training/improving-layouts/optimizing-layout.html [Accessed 5 Sept. 2015].

[Astronautix.com, n.d.] Encyclopedia Astronautica. *Navstar*. Available at: http://www.astronautix.com/project/navstar.htm [Accessed 27 Aug. 2015].

[Bowring, 1976] Bowring, B. R.. *Transformation from Spatial to Geographical Coordinates*. Survey Review, vol. 23, 1976.

[Bowring, 1985] Bowring, B. R.. *The Accuracy of Geodetic Latitude*. Survey Review, vol. 28, no. 218, 1976.

[Brady, 2008] Brady, Patrick (Google). *Anatomy & Physiology of an Android*. 2008. Available at: http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf [Accessed 18 Jul. 2015].

[Burtch, 2006] Burtch, Robert. *A Comparison of Methods Used in Rectangular to Geodetic Coordinate Transformations*. ACSM Annual Conference and Technology Exhibition, 2006.

[Dana, 2015] Dana, Peter H.. *Geodetic Datum Overview*. The Geographer's Craft Project, Department of Geography, The University of Colorado at Boulder, 2015. Available at: http://www.colorado.edu/geography/gcraft/notes/datum/datum.html [Accessed 30 Aug. 2015].

[Denti and Nurminen, 2013] Denti, Mattia, Nurminen, Jukka K.. *Performance and energy-efficiency of Scala on mobile devices*. Aalto University School of Science, 2013. Available at: http://cse.aalto.fi/en/midcom-serveattachmentguid-1e38756839a9dea875611e3b6c5bdaeedf47f937f93/denti_ngmast.pdf [Accessed 1 Sept. 2015].

[EGNOS, 2015] European Geostationary Navigation Overlay Service (EGNOS). *What is SBAS?*. 2015. Available at: http://egnos-portal.gsa.europa.eu/discover-egnos/about-egnos/what-sbas [Accessed 18 Aug. 2015].

[FAA, 2003] U.S. Department of Transportation Federal Aviation Administration - Airway Facilities Division. FAA Academy Training Manual - GPS concepts, Course 44221, 2003.

[FAA, 2015] U.S. Department of Transportation Federal Aviation Administration - Airway Facilities Division. *Navigation Programs - History - Satellite Navigation*. 2015. Available at: http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/history/satnav/ [Accessed 18 Aug. 2015].

[Featherstone and Claessens, 2008] Featherstone, W. E. and Claessens, S. J.. *Closed-Form Transformation Between Geodetic and Ellipsoidal Coordinates*. Studia Geophysica et Geodaetica, 2008.

[Frame and Coffey, 2014] Frame, Scott and Coffey, John W.. *A Comparison of Functional and Imperative Programming Techniques for Mathematical Software Development*. Journal of Systemics, Cybernetics and Informatics, vol.12, no. 2, 2014.

[Frysinger, 2015] Frysinger, Mike. (*N)compress version 4.2*. Available at: http://ncompress.sourceforge.net/ [Accessed 28 Aug. 2015].

[Gerdan and Deakin, 1999] Gerdan, George P. and Deakin, Rodney E.. *Transforming Cartesian coordinates X,Y,Z to Geographical coordinates φ, λ , h*. The Australian Surveyor, vol. 44, no. 1, 1999. Available at: http://www.researchgate.net/publication/254250446_Transforming_cartesian_coordinates_X_Y_Z_to_geographical_coordinates___h

[GitHub, 2015a] Barbeau, Sean. *Android GPS test program*. [source code shared via GitHub] Available at: https://github.com/barbeau/gpstest [Accessed 18 Jul. 2015].

[GitHub, 2015b] Gatis, Igor. *Compression in J2ME*. Available at: https://github.com/igorgatis/compress-j2me [Accessed 28 Aug. 2015].

[GNU, 2014] The GNU Multiple Precision Arithmetic Library (GMP). Available at: https://gmplib.org/ [Accessed 1 Sept. 2015].

[Google, 2015a] Google. *Google Earth projection*. 2015. Available at: https://support.google.com/earth/answer/148110?hl=en [Accessed 30 Aug. 2015].

[Google, 2015b] Google. *APIs Console Help*. 2015. Available at: https://developers.google.com/console/help/ [Accessed 24 Jul. 2015].

[Griffiths and Ray, 2008] Griffiths, Jake and Ray, Jim R.. *On the precision and accuracy of IGS orbits*. Springer-Verlag, 2008. Available at: http://www.ngs.noaa.gov/PUBS_LIB/OnThePrecisionAndAccuracyOfIGSOrbits.pdf

[Hegarty and Chatre, 2008] Hegarty, C.J.; Chatre, E.. *Evolution of the Global Navigation SatelliteSystem (GNSS)*. Proceedings of the IEEE , vol.96, no.12, 2008. Available at: http://ieeexplore.ieee.org/ieee_pilot/articles/96jproc12/jproc-CHegarty-2006090/article.html [Accessed 30 Aug. 2015].

[Heiskanen and Moritz, 1967] Heiskanen W.A., Moritz H.. *Physical Geodesy*. W.H. Freeman and Company, San Francisco, 1976.

[HowStuffWorks, 2005] Wilson, Tracy V.. *How GPS Phones Work*. 2005. Available at: http://electronics.howstuffworks.com/gps-phone.htm [Accessed 30 Aug. 2015].

[IGS, n.d.] International GNSS Service. Available at: http://www.igs.org/ [Accessed 18 Jul. 2015].

[IGS, 2009] International GNSS Service. *A GUIDE TO USING INTERNATIONAL GNSS SERVICE (IGS) PRODUCTS*. 2009. Available at: https://igscb.jpl.nasa.gov/components/usage.html [Accessed 18 Jul. 2015].

[IGS, 2015] International GNSS Service. [File from FTP site] Available at: ftp://ftp.igs.org/pub/product/1842/igu18425_12.sp3.Z [Accessed 18 Jul. 2015].

[iTunes, 2014] Radar, G. and Suzuki, T.. *GNSS Radar*. 2014. [App on iTunes App Store] Available at: https://itunes.apple.com/us/app/gnss-radar/id901597709 [Accessed 18 Jul. 2015].

[iTunes, 2015] *P-Track Satellite Viewer*. 2015. [App on iTunes App Store] Available at: https://itunes.apple.com/us/app/p-track-satellite-viewer/id473570351 [Accessed 18 Jul. 2015].

[JAXA, 2010] Mitsubishi Heavy Industries, Ltd. Japan Aerospace Exploration Agency (JAXA). *Launch Result of the First Quasi-Zenith Satellite 'MICHIBIKI' by H-IIA Launch Vehicle No. 18*. 2010. [Press release] Available at: http://global.jaxa.jp/press/2010/09/20100911_h2af18_e.html [Accessed 18 Aug. 2015].

[Kaplan, 1996] Kaplan, Elliott D. *Understanding Gps: Principles and Applications*. Artech House Mobile Communications Library. Boston: Artech House, 1996.

[Lee and Jeon, 2010] Lee, Sangchul and Jeon, Jae Wook. *Evaluating performance of Android platform using native C for embedded systems*. Control Automation and Systems (ICCAS), 2010 International Conference on, 2010.

[Navipedia, 2014] Navipedia. *WAAS Space Segment*. 2014. Available at: http://www.navipedia.net/index.php/WAAS_Space_Segment [Accessed 18 Aug. 2015].

[Nielson, 1994] Nielson, Jakob. *Usability Engineering*. San Francisco, Calif: Morgan Kaufmann Publishers, 1994. Available at: http://www.nngroup.com/articles/response-times-3-important-limits/ [Accessed 30 Aug. 2015].

[Newcastle, 2010] *IGS 2010 Newcastle upon Tyne Workshop - Recommendations*. 2010. Available at: https://igscb.jpl.nasa.gov/components/usage.html [Accessed 18 Jul. 2015].

[NGA, 1984] National Geospatial-Intelligence Agency. *World Geodetic System 1984*. 1984. Available at: http://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf [Accessed 18 Jul. 2015].

[NGA, 2014] National Geospatial-Intelligence Agency. *Gold Data v6.3 for Software Testing (UNCLASSIFIED)*. 2014. Available at: http://earth-info.nga.mil/GandG/coordsys/Conversion_Software/index.html [Accessed 18 Jul. 2015].

[NGS, 2003] National Geodetic Survey. *XYZ UTILITIES*. 2003. Available at: http://www.ngs.noaa.gov/TOOLS/XYZ/xyz.html [Accessed 18 Jul. 2015].

[R, 2015] R. Available at: https://www.r-project.org/ [Accessed 3 Sept. 2015].

[SciPy, 2015] Scientific Python. Available at: http://www.scipy.org/ [Accessed 1 Sept. 2015].

[Smart Insights, 2015] Smart Insights. *Mobile marketing statistics 2015*. Available at: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ [Accessed 18 Jul. 2015].

[SQLite, 2015] SQLite. *SQLite Android Bindings Documentation*. 2015. Available at: https://www.sqlite.org/android/doc/trunk/www/index.wiki [Accessed 30 Aug. 2015].

[Steiniger et al., n.d.] Steiniger, Stefan, Neun, Moritz and Edwardes, Alistair. *Foundations of Location Based Services*. University of Zurich, n.d.. Available at: http://sourceforge.net/projects/jump-pilot/files/w_other_freegis_documents/articles/lbs_lecturenotes_steinigeretal2006.pdf/download [Accessed 28 Aug. 2015].

[Torge, 2001] Torge, Wolfgang. *Geodesy*. Walter de Gruyter GmbH & Co., 2001.

[Tsui, 2000] Tsui, James Bao-yen. *Fundamentals of Global Positioning System Receivers: A Software Approach*. Wiley Series in Microwave and Optical Engineering. New York: Wiley, 2000.

[United States, 1996] United States. Air Force. Navstar Global Positioning System Program Office & Navstar Seminars. *NAVSTAR GPS user equipment : introduction (Public release version)*. Navtech Seminars & Navtech Book and Software Store, 1996. Available at: http://www.navcen.uscg.gov/pubs/gps/gpsuser/gpsuser.pdf [Accessed 28 Aug. 2015].

[USGS, 2013] United States Geological Survey. *Teaching About and Using Coordinate Systems*, 2013. Available at: http://education.usgs.gov/lessons/coordinatesystems.pdf [Accessed 30 Aug. 2015].

[Wang et al., 2008] Wang, Shu, Min, Jungwon and Yi, Byung K.. *Location Based Services for Mobiles: Technologies and Standards*. IEEE International Conference on Communication (ICC), 2008. Available at: http://to.swang.googlepages.com/ICC2008LBSforMobilessimplifiedR2.pdf [Accessed 28 Aug. 2015].

[Wasserman, 2010] Wasserman, Anthony I.. *Software engineering issues for mobile application development*. Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER '10), 2010. Available at: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1040&context=silicon_valley [Accessed 28 Aug. 2015].

[Yunnanexplorer, n.d.] Yunnanexplorer.com. Available at: http://www.yunnanexplorer.com/download/androidapps/ [Accessed 18 Jul. 2015].

[Zhu, 1994] J., Zhu. *Conversion of Earth-centered Earth-fixed coordinates to geodetic coordinates*. Aerospace and Electronic Systems, IEEE Transactions on, vol. 30, 1994.

Appendix 1 – XYZ to LLH Coordinate Conversion Code

Calculation for Longitude

```
public static double calculateLongitude(VehicleObj v) {
  double x = v.xcoordinate;
  double y = v.ycoordinate;
  double z = v.zcoordinate;
  double longitude = 0.0;
  if (x >= 0.0) {
    longitude = Math.atan(y / x);
  } else if (x < 0 && y >= 0.0) {
    longitude = Math.atan(y / x) + Math.PI;
  } else {
    longitude = Math.atan(y / x) - Math.PI;
  }
  //wrap longitude at the Prime Meridian
  longitude = longitude
    + 2.0 * Math.PI * ((longitude < 0.0) ? 1.0 : 0.0)
    - 2.0 * Math.PI * ((longitude > 2.0 * Math.PI) ? 1.0 : 0.0);
  longitude = Math.toDegrees(longitude);
  return longitude;
}
```

Table A1.1 An example code for calculation longitude

Calculation for Latitude: Kaplan's 15-Step Method

```
public static VehicleObj xyz2geo_kaplan(VehicleObj v) {
  //Parameters from WGS84 datum
  double a = 6378137.0;
  double b = 6356752.3142;

  // eccentricity
  double e = Math.sqrt(1.0 - (Math.pow(b, 2.0)/Math.pow(a, 2.0)));
  // second eccentricity, e prime
  double ep = (a/b) * e;

  double x = v.xcoordinate;
  double y = v.ycoordinate;
  double z = v.zcoordinate;

  double r = Math.sqrt(Math.pow(x, 2.0) + Math.pow(y, 2.0));
  double E2 = Math.pow(a, 2.0) - Math.pow(b, 2.0);
  double F = 54.0 * Math.pow(b, 2.0) * Math.pow(z, 2.0);
  double G = Math.pow(r, 2.0) + (1.0 - Math.pow(e, 2.0)) * Math.pow(z, 2.0) -
Math.pow(e, 2.0) * E2;
  double c = (Math.pow(e, 4.0) * F * Math.pow(r, 2.0)) / Math.pow(G, 2.0);
  double s = Math.cbrt(1.0 + c + Math.sqrt(Math.pow(c, 2.0) + 2.0 * c));
  double P = F / (3.0 * Math.pow((s + (1.0/s) + 1.0), 2.0) * Math.pow(G, 2.0));
  double Q = Math.sqrt(1.0 + (2.0 * Math.pow(e, 4.0) * P));

  double r0a = (P * Math.pow(e, 2.0) * r) / (1.0 + Q);
  double r0b = (1.0/2.0) * Math.pow(a, 2.0) * (1.0 + (1.0/Q));
  double r0c = (P * (1.0 - Math.pow(e, 2.0) * Math.pow(z, 2.0))) / (Q * (1.0 + Q));
  double r0d = (1.0/2.0) * P * Math.pow(r, 2.0);
  double r0 = -r0a + Math.sqrt(r0b - r0c - r0d);

  double U = Math.sqrt(Math.pow(r - (Math.pow(e, 2.0) * r0), 2.0) + Math.pow(z, 2.0));
  double V = Math.sqrt(Math.pow(r - (Math.pow(e, 2.0) * r0), 2.0) + (1.0 - Math.pow(e,
2.0)) * Math.pow(z, 2.0));
  double z0 = (Math.pow(b, 2.0) * z) / (a * V);
  double altitude = U * (1.0 - (Math.pow(b, 2.0)/a * V));
  double latitude = Math.atan((z + Math.pow(ep, 2.0) * z0) / r);

  v.latitude_kaplan = PositionCalculations.RadtoDeg(latitude);
  v.altitude_kaplan = altitude;

  return v;
}
```

Table A1.2 An example code for calculation latitude with Kaplan's method

Calculation for Latitude: Hirvonen & Moritz's Method

```
public static VehicleObj xyz2geo_hirvonenMoritz(VehicleObj v) {
  //Parameters from WGS84 datum
  double a = 6378137.0;
  double b = 6356752.3142;

  double precision = 0.000000000001;
  double e2 = (a*a -b*b)/(a*a);

  double E = v.xcoordinate;
  double F = v.ycoordinate;
  double G = v.zcoordinate;

  double p = Math.sqrt(Math.pow(E,2.0) + Math.pow(F,2.0));

  //first approximation:
  double lat1 = Math.atan((G/p) * (1.0 + (e2/(1.0-e2))));
  double N = a / (Math.sqrt(1 - (e2 * Math.pow((Math.sin(lat1)), 2.0))));

  //iterate lat1 until change in lat is insignificant:
  double latitude = Math.atan((G / p) * (1 + ((e2 * N * Math.sin(lat1)) / G)));
  int numOfIts = 0.0;
  while(
   Math.abs(Math.abs(Math.toDegrees(lat1)) - Math.abs(Math.toDegrees(latitude)))
   > precision
  ) {
    lat1 = latitude;
    double newN = a / (Math.sqrt(1.0 - (e2 * Math.pow((Math.sin(lat1)), 2))));
    if(!Double.isNaN(newN)) {
      N = newN;
      latitude = Math.atan2(G / p, (1.0 + (e2 * N * Math.sin(lat1) / G)));
    } else {
      lat1 = latitude;
    }
    numOfIts++;
  }//end iterate

  double altitude = (p / (Math.cos(lat1))) - N;

  v.latitude_hirvonenMoritz = Math.toDegrees(latitude);
  v.altitude_hirvonenMoritz = altitude;
  v.numOfIterations_hirvonenMoritz = numOfIts;
  return v;
}
```

Table A1.3 An example code for calculation latitude with Hirvonen & Moritz's method

Calculation for Latitude: Torge's (Heiskanen and Moritz's) Method

```
public static VehicleObj xyz2geo_torge(VehicleObj v) {
  //Parameters from WGS84 datum
  double a = 6378137.0;
  double b = 6356752.3142;

  double precision = 0.000000000001;
  double e2 = (a*a -b*b)/(a*a);

  double E = v.xcoordinate;
  double F = v.ycoordinate;
  double G = v.zcoordinate;

  double p = Math.sqrt(Math.pow(E,2.0) + Math.pow(F,2.0));

  //first approximation:
  double lat1 = Math.atan((G/p) * (1.0/(1.0-e2)));
  double N = a / (Math.sqrt(1 - (e2 * Math.pow((Math.sin(lat1)), 2.0))));
  double altitude = (p / (Math.cos(lat1))) - N;

  //iterate lat1 until change in lat is insignificant:
  double latitude = Math.atan((G / p) * (1.0 / (1.0-(e2 * N/(N+altitude)))));
  int numOfIts = 0.0;
  while(
   Math.abs(Math.abs(Math.toDegrees(lat1)) - Math.abs(Math.toDegrees(latitude)))
   > precision
  ) {
    lat1 = latitude;
    double newN = a / (Math.sqrt(1.0 - (e2 * Math.pow((Math.sin(lat1)), 2.0))));
    altitude = (p / (Math.cos(lat1))) - N;
    if(!Double.isNaN(newN)) {
      N = newN;
      latitude = Math.atan((G / p) * (1.0 / (1.0-(e2 * N/(N+altitude)))));
    } else {
      lat1 = latitude;
    }
    numOfIts++;
  }//end iterate

  v.latitude_torge = Math.toDegrees (latitude);
  v.altitude_torge = altitude;
  v.numOfIterations_torge = numOfIts;

  return v;
}
```

Table A1.4 An example code for calculation latitude with Torge's method

Calculation for Latitude: Bowring's Method

Note that Bowring's method is an iterative one. However, the experimentations for this thesis has consistently shown that only one iteration is required to accomplish a result within 0.00001 change in the value of φ. Therefore, the single iteration version shown below has been implemented in the app.

```
public static VehicleObj xyz2geo_bowring(VehicleObj v) {
  //Parameters from WGS84 datum
  double a = 6378137.0;
  double b = 6356752.3142;

  double x = v.xcoordinate;
  double y = v.ycoordinate;
  double z = v.zcoordinate;

  // 1st eccentricity squared
  double E1 = (Math.pow(a, 2.0) - Math.pow(b, 2.0)) / Math.pow(a, 2.0);
  // 2nd eccentricity squared
  double E2 = (Math.pow(a, 2.0) - Math.pow(b, 2.0)) / Math.pow(b, 2.0);
  // distance from minor axis
  double p = Math.sqrt(Math.pow(x, 2.0) + Math.pow(y, 2.0));
  // polar radius
  double R = Math.sqrt(Math.pow(p, 2.0) + Math.pow(z, 2.0));

  // parametric latitude (Bowring eqn 17, replacing tanb)
  double tanb = (b*z)/(a*p) * (1.0+E2*b/R);
  double sinb = tanb / Math.sqrt(1+Math.pow(tanb, 2.0));
  double cosb = sinb / tanb;

  // geodetic latitude (Bowring eqn 18)
  double latitude = Math.atan2(z+E2*b*Math.pow(sinb,3.0), p - E1*a*Math.pow(cosb,3.0));

  // height above ellipsoid (Bowring eqn 7) [not currently used]
  double sinlat = Math.sin(latitude);
  double coslat = Math.cos(latitude);
  // length of the normal terminated by the minor axis
  double norm = a*Math.sqrt(1-E1*Math.pow(sinlat, 2.0));
  double altitude = (p*coslat) + (z*sinlat) - (Math.pow(a, 2.0)/norm);

  v.latitude_bowring = PositionCalculations.RadtoDeg(latitude);
  v.altitude_bowring = altitude;

  return v;
}
```

Table A1.6 An example code for calculation latitude with Bowring's method

Data Extraction with Google Code

The historic data for the satellite location is stored on the IGS website. The data files are contained in .sp3 files, and compressed into .Z files using UNIX compress (LZC) algorithm.

```
URL url = new URL(IGS_website);
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("GET");
InputStream inputstream = connection.getInputStream();
FileOutputStream outputstream = new FileOutputStream(local_filename);
LZCInputStream lzc = new LZCInputStream(inputstream);
lzc.uncompress(inputstream, outputstream);
inputstream.close();
```

Table A2.1 An example code of data extraction with Google Code

AndroidManifest.xml

The API Key obtained from the Google API Console needs to be inserted into the AndroidManifest.xml file. In addition, several user permissions need to be included as well. Lastly, it also needs the version of the google play services used for the app.

Google Maps-specific variables are highlighted in a sample AndroidManifest.xml below.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.android.gpstest.testGoogleMapsAPI " >
  <uses-sdk
    android:minSdkVersion="11"
    android:targetSdkVersion="21" />

  <uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <meta-data
      android:name="com.google.android.gms.version"
      android:value="@integer/google_play_services_version" />

    <meta-data
      android:name="com.google.android.maps.v2.API_KEY"
      android:value="AIzaSyDYL1Oo4e4Pu6p1JLYtj6M2tGYk3UNuaE8" />

    <activity
      android:name=".MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Table A3.1 A sample AndroidManifest.xml

build.gradle

The app needs to add a new build rule in `build.gradle`, under dependencies for the latest version of `play-services`. This is highlighted in a sample `build.gradle` file below.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"

    defaultConfig {
        applicationId "com.android.gpstest.testGoogleMapsAPI "
        minSdkVersion 11
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles   getDefaultProguardFile('proguard-android.txt'),   'proguard-
rules.pro'
        }
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile files('libs/google-play-services.jar')
    compile files('libs/joda-time-2.7.jar')
    compile 'com.android.support:support-v4:21.0.3'
    compile 'com.google.android.gms:play-services:6.5.87'
    compile 'com.android.support:appcompat-v7:21.0.3'
}
```

Table A3.1 A sample `AndroidManifest.xml`

Sample `activity_main.xml`

The simplest way to add a map to an application is to specify it in the user interface layout XML file for an activity.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
  tools:context=".MainActivity">

    <fragment
        android:id="@+id/map"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical"
        class="com.google.android.gms.maps.MapFragment"/>

</RelativeLayout>
```

Table A3.1 Sample `MapFragment` instance added to the `activity_main.xml`

Kaplan's Method (with `double` data type)

Kaplan's method was selected for the precision benchmark of the impact of using Java floating point data types. This is because Kaplan's method uses only one trigonometric function, which means it does not require conversion between `double` and `BigDecimal` in order to calculate the trigonometric functions. In the first test, the calculations used the `double` data type.

```
public static VehicleObj kaplan(VehicleObj v) {
  //Parameters from WGS84 datum
  double a = 6378137.0;
  double b = 6356752.3142;

  // eccentricity
  double e = Math.sqrt(1.0 - (Math.pow(b, 2.0)/Math.pow(a, 2.0)));

  //flattening
  double f = 1.0 - (b/a);
  double ep = Math.sqrt((Math.pow(a, 2.0)/Math.pow(b, 2.0)) -1.0);

  double x = v.xcoordinate;
  double y = v.ycoordinate;
  double z = v.zcoordinate;

  double r = Math.sqrt(Math.pow(x, 2.0) + Math.pow(y, 2.0));
  double E2 = Math.pow(a, 2.0) - Math.pow(b, 2.0);
  double F = 54.0 * Math.pow(b, 2.0) * Math.pow(z, 2.0);
  double G = Math.pow(r, 2.0) + (1.0 - Math.pow(e, 2.0)) * Math.pow(z, 2.0) -
Math.pow(e, 2.0) * E2;
  double c = (Math.pow(e, 4.0) * F * Math.pow(r, 2.0)) / Math.pow(G, 2.0);
  double s = Math.cbrt(1.0 + c + Math.sqrt(Math.pow(c, 2.0) + 2.0 * c));
  double P = F / (3.0 * Math.pow((s + (1.0/s) + 1.0), 2.0) * Math.pow(G, 2.0));
  double Q = Math.sqrt(1.0 + (2.0 * Math.pow(e, 4.0) * P));

  double r0a = (P * Math.pow(e, 2.0) * r) / (1.0 + Q);
  double r0b = (1.0/2.0) * Math.pow(a, 2.0) * (1.0 + (1.0/Q));
  double r0c = (P * (1.0 - Math.pow(e, 2.0) * Math.pow(z, 2.0))) / (Q * (1.0 + Q));
  double r0d = (1.0/2.0) * P * Math.pow(r, 2.0);
  double r0 = -r0a + Math.sqrt(r0b - r0c - r0d);

  double U = Math.sqrt(Math.pow(r - (Math.pow(e, 2.0) * r0), 2.0) + Math.pow(z, 2.0));
  double V = Math.sqrt(Math.pow(r - (Math.pow(e, 2.0) * r0), 2.0) + (1.0 - Math.pow(e,
2.0)) * Math.pow(z, 2.0));
  double z0 = (Math.pow(b, 2.0) * z) / (a * V);
  double altitude = U * (1.0 - (Math.pow(b, 2.0)/a * V));
  double latitude = Math.atan((z + Math.pow(ep, 2.0) * z0) / r);
  v.latitude_kaplan = PositionCalculations.RadtoDeg(latitude);

  return v;
}
```

Table A4.1 Calculation of Kaplan's method with the `double` data type

Kaplan's Method (with `BigDecimal` data type)

In the second test, the calculations used the `BigDecimal` data type.

```
public static void kaplanBD(VehicleObj v) {
  BigDecimal BD1neg = new BigDecimal(-1);
  BigDecimal BD1 = new BigDecimal(1);
  BigDecimal BD2 = new BigDecimal(2);
  BigDecimal BD3 = new BigDecimal(3);
  BigDecimal BD54 = new BigDecimal(54);

  BigDecimal x = new BigDecimal(v.xcoordinate);
  BigDecimal y = new BigDecimal(v.ycoordinate);
  BigDecimal z = new BigDecimal(v.zcoordinate);
  BigDecimal a = new BigDecimal(6378137.0); //Equatorial cross-section radius in km /
semimajor axis (mean equatorial radius) [km]
  BigDecimal b = new BigDecimal(6356752.3142); //Polar cross-section radius in km /
semiminor axis [km]

  BigDecimal z2 = z.multiply(z, MathContext.DECIMAL128);
  BigDecimal a2 = a.multiply(a, MathContext.DECIMAL128);
  BigDecimal b2 = b.multiply(b, MathContext.DECIMAL128);
  BigDecimal e = takeRoot(BD1.subtract((b2).divide(a2, MathContext.DECIMAL128)), 2);
  BigDecimal e2 = e.multiply(e, MathContext.DECIMAL128);
  BigDecimal e4 = e2.multiply(e2, MathContext.DECIMAL128);
  BigDecimal ep = (a.divide(b, MathContext.DECIMAL128)).multiply(e,
MathContext.DECIMAL128);
  BigDecimal ep2 = ep.multiply(ep, MathContext.DECIMAL128);
  BigDecimal oneminuse2 = BD1.subtract(e2);

  //r
  BigDecimal r2 = (x.multiply(x, MathContext.DECIMAL128)).add(y.multiply(y,
MathContext.DECIMAL128));
  BigDecimal r = takeRoot((x.multiply(x, MathContext.DECIMAL128)).add(y.multiply(y,
MathContext.DECIMAL128)), 2);

  //E2
  BigDecimal E2 = a2.subtract(b2);

  //F
  BigDecimal F = BD54.multiply(b2, MathContext.DECIMAL128).multiply(z2,
MathContext.DECIMAL128);

  //G
  BigDecimal G = r2.add(oneminuse2.multiply(z2,
MathContext.DECIMAL128)).subtract(e2.multiply(E2, MathContext.DECIMAL128));
  BigDecimal G2 = G.multiply(G, MathContext.DECIMAL128);
  BigDecimal G3 = G.multiply(G, MathContext.DECIMAL128).multiply(G,
MathContext.DECIMAL128);

  //c
  BigDecimal c = e2.multiply(e2, MathContext.DECIMAL128).multiply(F,
MathContext.DECIMAL128).multiply(r2, MathContext.DECIMAL128).divide(G3,
MathContext.DECIMAL128);
  BigDecimal c2 = c.multiply(c, MathContext.DECIMAL128);

  //s
  BigDecimal s = takeRoot(BD1.add(c).add(takeRoot((c2.add(BD2.multiply(c,
MathContext.DECIMAL128))), 2)), 3);

  //P
  BigDecimal Pppp = s.add(BD1.divide(s, MathContext.DECIMAL128)).add(BD1);
  BigDecimal Ppp = Pppp.multiply(Pppp, MathContext.DECIMAL128);
```

```
  BigDecimal Pp = BD3.multiply(Ppp, MathContext.DECIMAL128).multiply(G2,
MathContext.DECIMAL128);
  BigDecimal P = F.divide((Pp), MathContext.DECIMAL128);

  //Q
  BigDecimal Q = takeRoot((BD1.add(BD2.multiply(e4, MathContext.DECIMAL128).multiply(P,
MathContext.DECIMAL128))), 2);

  //r0
  BigDecimal r0a = (P.multiply(e2, MathContext.DECIMAL128).multiply(r,
MathContext.DECIMAL128)).divide((BD1.add(Q)), MathContext.DECIMAL128);
  BigDecimal r0b = ((BD1.divide(BD2)).multiply(a2,
MathContext.DECIMAL128)).multiply((BD1.add((BD1.divide(Q, MathContext.DECIMAL128)))),
MathContext.DECIMAL128);
  BigDecimal r0c = (P.multiply(oneminuse2, MathContext.DECIMAL128).multiply(z2,
MathContext.DECIMAL128)).divide((Q.multiply((BD1.add(Q)), MathContext.DECIMAL128)),
MathContext.DECIMAL128);
  BigDecimal r0d = BD1.divide(BD2, MathContext.DECIMAL128).multiply(P,
MathContext.DECIMAL128).multiply(r2, MathContext.DECIMAL128);
  BigDecimal r0 = (BD1neg.multiply(r0a,
MathContext.DECIMAL128)).add(takeRoot((r0b.subtract(r0c).subtract(r0d)), 2));

  //U
  BigDecimal Ua = r.subtract(e2.multiply(r0, MathContext.DECIMAL128));
  BigDecimal Ub = Ua.multiply(Ua, MathContext.DECIMAL128);
  BigDecimal U = takeRoot((Ub.add(z2)), 2);

  //V
  BigDecimal V = takeRoot((Ub.add(oneminuse2.multiply(z2, MathContext.DECIMAL128))),
2);

  //z0
  BigDecimal z0 = (b2.multiply(z, MathContext.DECIMAL128)).divide(a.multiply(V,
MathContext.DECIMAL128), MathContext.DECIMAL128);

  //altitude
  BigDecimal altitude = U.multiply((BD1.subtract(b2.divide((a.multiply(V,
MathContext.DECIMAL128)), MathContext.DECIMAL128))), MathContext.DECIMAL128);

  //latitude:
  BigDecimal la = (z.add(ep2.multiply(z0, MathContext.DECIMAL128))).divide(r,
MathContext.DECIMAL128);
  double lb = la.doubleValue();
  double latitude = Math.atan(lb);
  double latitude_kaplan = PositionCalculations.RadtoDeg(latitude);
}
```

Table A4.2 Calculation of Kaplan's method with the `BigDecimal` data type

Kaplan's Method (using R)

In the third test, the calculations were done in R.

```
xyz2geo <- function(xyz){

  # CONSTANTS
  # semimajor axis (mean equatorial radius) [m]
  a = 6378137;
  # semiminor axis [m]
  b = 6356752.3142;
  # eccentricity
  e = sqrt(1 - b^2/a^2);
  # second eccentricity
  ep = 0.0820944379496;

  # Evaluation of Geodetic height and latitude
  r = sqrt(sum(xyz[1:2]^2));
  E2 = a^2 - b^2;
  f = 54*b^2*xyz[3]^2;
  G = r^2 + (1 - e^2)*xyz[3]^2 - e^2*E2;
  c = e^4*f*r^2/G^3;
  s = (1 + c + sqrt(c^2+2*c))^(1/3);
  P = f/(3*(s + 1/s + 1)^2*G^2);
  Q = sqrt(1 + 2*e^4*P);
  r0 = -P*e^2*r/(1 + Q) + sqrt(0.5*a^2*(1+1/Q) - P*(1 - e^2)*xyz[3]^2/(Q*(1 + Q)) -
0.5*P*r^2);
  U = sqrt((r - e^2*r0)^2 + xyz[3]^2);
  V = sqrt((r - e^2*r0)^2 + (1 - e^2)*xyz[3]^2);
  z0 = b^2*xyz[3]/(a*V);
  llh[3] = U*(1-b^2/(a*V));
  llh[1] = atan2((xyz[3]+ep^2*z0),r);

  if( xyz[1] >= 0){
    llh[2] = atan(xyz[2]/xyz[1]);
  } else if(xyz[1] < 0 && xyz[2] >= 0 ){
    llh[2] = atan(xyz[2]/xyz[1]) + pi;
  } else {
    llh[2] = atan(xyz[2]/xyz[1]) - pi;
  }
  # wrap longitude
  llh[2] = llh[2] + 2*pi*(llh[2] < 0) - 2*pi*(llh[2] > 2*pi);

  return(llh);
}
```

Table A4.3 Calculation of Kaplan's method using R